UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

ESTUDIO DE LA INTEGRACIÓN DE M2M Y SISTEMAS DE INFORMACIÓN

Autor: David Salas Guadalajara

Tutor: Daniel Díaz Sánchez

5 de julio de 2016

Agradecimientos

A mi tutor Daniel, por ayudarme durante todo el desarrollo del proyecto, guiarme y aconsejarme en cada momento.

A mis padres y mi hermano, por haberme ayudado, tanto en el proyecto como a lo largo de la carrera y haberme dado fuerzas para seguir hasta el final.

A Mariam, por animarme en todo momento, por darme ilusión de seguir y por ser mi compañera de viaje.

Resumen

Las telecomunicaciones están en pleno auge. El sector de las TIC ha revolucionado el mundo desde la década de los 70. Los avances y el consumo han crecido de forma exponencial desde sus inicios y una de las tendencias actuales es el Internet de las Cosas (IoT): Todos los objetos y dispositivos conectados entre sí por internet, enviándose millones de mensajes para transferir información, información que se utilizará para algún tipo de aplicación.

Se prevé que en 2020 haya 50 millones de dispositivos conectados. Se pretende que haya aplicaciones de diversos tipos, tanto para conducción autónoma, como información, seguridad, salud, etc.

A la vista de este panorama, hay que plantear soluciones al problema de la múltiple conexión de dispositivos. Es necesario que haya elementos que permitan interconectarlos, que se comuniquen en diferentes protocolos, que permita, posiblemente, millones de conexiones con dispositivos sin problemas de procesamiento o almacenaje, o incluso, que sea escalable. Por tanto, se ha decidido que en este proyecto se diseñará ese elemento que es fundamental para el funcionamiento de las tecnologías antes mencionadas. Lo que se diseñará será un Gateway preparado para conectarse con diferentes elementos, intercomunicarlo, traducir entre protocolos, y un sistema de escalabilidad para mejorar las propiedades cuando aumente la cantidad de dispositivos.

Abstract

Telecommunications are booming. The ICT sector has revolutionized the world since the early 70 Advances and consumption have grown exponentially since its inception and one of the current trends is the Internet of Things (IoT): All objects and devices connected by internet, sending millions of messages to transfer information, information to be used for any type of application.

It is expected to be 50 billion connected devices by 2020. It is intended to be applications of various types: autonomous driving, information, safety, health, etc.

In view of this scenario, should propose solutions to the problem of connecting multiple devices. There needs to be elements to interconnect them, communicate in different protocols, allowing possibly millions of connections with devices without processing or storage problems, or even that is scalable. Therefore, it was decided that in this project, that element that is critical to the operation of the above technologies, will be designed. What was designed to be a Gateway ready to connect with different elements, communicate it, translate between protocols, and system scalability to improve the properties when you increase the number of devices.

Índice general

Índice de figuras

Capítulo 1: Introducción

1.1 Motivación

El crecimiento de la tecnología es algo conocido. Es obvio decir que actualmente la telefonía es algo universalmente extendido, y cada vez es más frecuente el uso de dispositivos tales como tabletas o smartTVs.

Actualmente hay aproximadamente 7.2 miles de millones de personas en la tierra (de las cuales no todas tienen acceso a internet o a las tecnologías actuales), y la cantidad actual de dispositivos conectados es de 25 mil millones. Según Cisco[1] prevé, en 2020 haya el doble de dispositivos, es decir, habrá 6.58 dispositivos[1] conectados por persona.

Es necesaria una mejora en las comunicaciones puesto que en poco tiempo veremos múltiples aplicaciones con muchísimos dispositivos conectándose entre sí, transmitiéndose información.

En los próximos años habrá servicios que requieran altos niveles de anchos de bandas. Internet of Things es uno de los mayores propulsores de los sistemas M2M[2] dado que para el funcionamiento de las aplicaciones, será necesaria una total independencia de las personas.

Algunos de los principales objetivos en el sector de IoT son:

- Smart Grid: transmisión de medidas en la producción de servicios
- Monitorización: de temperatura, luz, humedad y su regulación
- Servicios de emergencia: como alarmas, que requerirán de una alta prioridad
- eHealth: cuidado de salud con dispositivos comunicados, requerirán perioricidad
- información y navegación: comunicaciones aleatorias

[1] D. Evans, The internet of things: How the next evolution of the internet is changing everything, CISCO white paper 1.

[2] Machine-to-Machine (M2M) communications: A survey, Journal of Network and Computer Applications Volume 66, May 2016, Pages 83–105, http://www.sciencedirect.com/science/article/pii/S1084804516000990

- otras aplicaciones que puedan ser elásticas, o a tiempo real...

Por una parte habrá aplicaciones que requieran protocolos orientados a conexión, como TCP. Estas aplicaciones necesitarán transmitir de forma segura paquetes para que sus destinos los reciban con éxito, pero este tipo de conexiones requieren reserva de recursos, que pueden ser muchos cuando se trata de aplicaciones para muchos dispositivos interconectados. También se puede producir congestión por mantener demasiadas conexiones. Por otra, otras aplicaciones requerirán una comunicación no tan estricta, bien sea porque la información a transmitir no es tan importante, o porque no necesite comunicarse constantemente. Esto puede realizarse con protocolos como UDP, que envían mensajes sin asegurarse su recepción, sin que tenga que haber alguien obligatoriamente recibiéndolos. Es una comunicación menos fiable pero más ligera.

Una aplicación de eHealth puede que necesite el primer tipo de conexión, dado que una pequeña variación en alguna medida en una persona de salud débil, puede ser muy importante para prevenir alguna enfermedad. No se pueden dejar pasar esas medidas. En cambio una aplicación de Crowd Sensing puede que sus medidas sean prescindibles y no importe que se pierdan, por lo que le convendría una comunicación del segundo tipo.

Categoría	**2014**	**2015**	**2016**	**2020**
Consumidor	2,277	3,023	4,024	13,509
Negocios en general	632	815	1,092	4,408
Negocios verticales	898	1,065	1,276	2,880
Total	**3,807**	**4,902**	**6,392**	**20,797**

Figura 1: Dispositivos de IoT conectados ordenados por sector[3]

Como podemos ver, hay una gran cantidad de futuras aplicaciones, y algunas puede que no necesiten reserva de recursos, pero otras sí lo necesitarán. Independientemente del carácter de las aplicaciones, se

[3] Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015 http://www.gartner.com/newsroom/id/3165317

puede confirmar que necesitarán un gran número de comunicaciones, dado que su característica más importante es la intercomunicación entre un gran número de dispositivos.

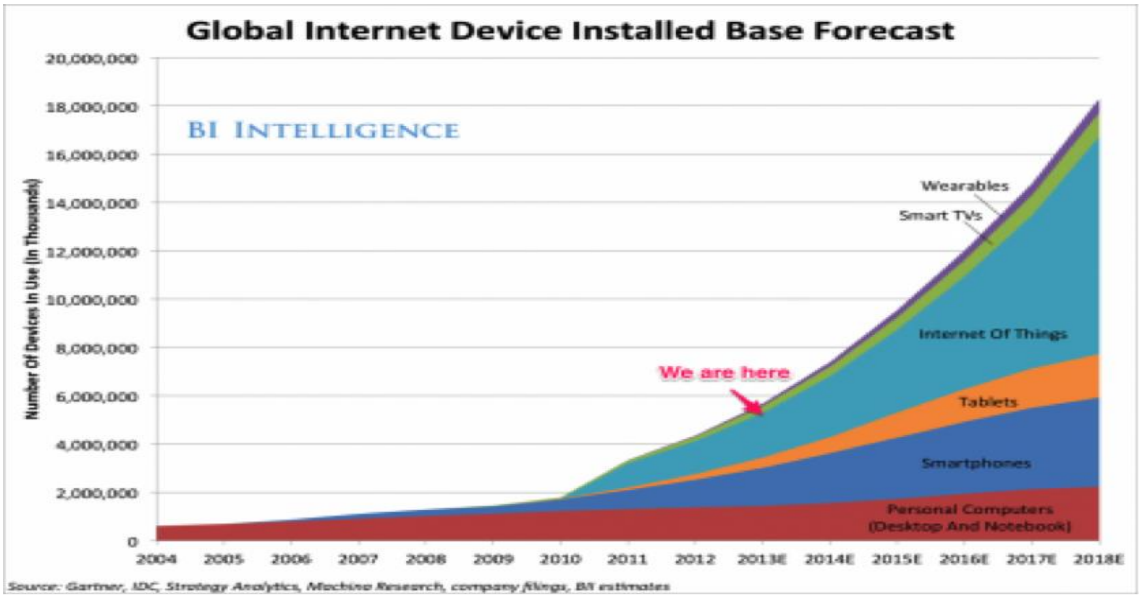

Figura 2: Crecimiento de IoT con respecto a otros sectores[4]

Por todas estas razones es importante desarrollar sistemas que permitan la intercomunicación entre los dispositivos, que puedan encaminar mensajes, duplicarlos, comunicarse en diferentes protocolos, traducir mensajes de un protocolo a otro e incluso almacenar información. También es importante que estos elementos tengan propiedades como la seguridad y la escalabilidad.

• 1.2 Objetivos:

Debido a estas necesidades, para este proyecto se ha decidido diseñar un Gateway para tecnologías M2M, que sea capaz de soportar múltiples conexiones, entrantes o salientes, y que sea capaz de realizar las funciones mencionadas anteriormente. Lo que se desea esque puedan

[4] Gartner ADC, Strategy Analytics, Machine Researching, company filings, BII estimates.

conectarse entre sí muchos elementos y que puedan realizarse agrupaciones, de tal forma que podamos elegir para un mismo emisor varios receptores o viceversa, que permita el intercambio de mensajes entre diferentes protocolos y la escalabilidad, ejecutándose un mismo programa en diferentes máquinas y que su localización sea algo transparente a la hora de su ejecución.

Todo esto será posible principalmente gracias a los sistemas de actores de Akka y las librerías de Apache Camel. Como veremos en el apartado 2 (Estado del arte), el sistema de actores de Akka es un conjunto de librerías que proporciona unos objetos llamados actores, los cuales representarán a cualquier tipo de objeto y una vez creados, no se podrán modificar. Se tratarán mediante referencias con el fin de proteger la información que contienen dichos actores, y se comunicarán entre sí mediante mensajes.

Por otra parte Apache Camel proporciona un sistema de productores y consumidores, estos serán los encargados de producir y de recibir mensajes respectivamente del exterior. Cada consumidor y cada productor tendrá una dirección IP y puerto en los que escuchar o a los que enviar los mensajes. Ademas, Camel permite que dichos mensajes puedan enviarse en diferentes protocolos. En los componentes de Camel se pueden encontrar una gran cantidad de protocolos sobre los cuales se permite trabajar. Es muy fácil con estas librerías crear un mensaje de SIP, o un mensaje de HDFS, sin tener que desarrollar campo por campo dicho mensaje.

Gracias a estas herramientas se podrá realizar el objetivo del proyecto. Ambas simplifican mucho la programación, dado que el diseño de un Gateway que opere en diferentes protocolos, que sea escalable, seguro, asíncrono, que permita agrupaciones es muy complicado si se crea desde 0. Afortunadamente todas estas opciones mencionadas están ya desarrolladas y solo se tiene que implementar lo que ya está creado.

Por último, cabe mencionar que para el proyecto no se ha desarrollado el Gateway en su totalidad, sino que el objetivo es demostrar que con estas herramientas es posible crear dicho Gateway de una forma sencilla, que

pueda tener estas propiedades, y crear una pequeña aplicación que sea ejemplo de las capacidades de estas herramientas a pequeña escala. En el apartado 4 (implementación) se verá con más exactitud cuáles de las funciones del proyecto son las que se podrán implementar y cuáles, aun sabiendo que son posibles, no se han desarrollado por falta de recursos tanto económicos como temporales.

En definitiva, ¿qué es lo que se quiere conseguir? Lo que se busca en este proyecto es diseñar una arquitectura software que pueda demostrar que con Akka y Camel como herramientas y java como lenguaje de programación, se pueda crear un software sencillo que al ejecutarse, la máquina donde esté funcionando se comporte como un elemento al que puedan acceder múltiples dispositivos y se puedan comunicar entre sí, traducirse protocolos, etcétera;

• 1.3 Contenido de la memoria

Vista ya la motivación que ha llevado a realizar este proyecto, y el objetivo que tiene, se va a pasar a detallar los apartados que habrá en el documento de la memoria.

En primer lugar estará el estado del arte, apartado en el cual se detalle en profundidad todas las tecnologías empleadas para el proyecto, que no se hayan visto a lo largo de la carrera. Es obvio que explicar en dicho apartado cómo funciona Java o eclipse es inútil, se sobreentiende que a lo largo de la ingeniería se han visto estas herramientas y por tanto lo que se desarrolla en este apartado son las tecnologías que no tienen por qué verse a lo largo de la carrera, y las cuales son fundamentales para el desarrollo del proyecto.

A continuación de este apartado, se verá el diseño en general del proyecto, con todos los elementos que se necesitarían para cumplir el objetivo, cómo se podría realizar cada parte, pero dejando claro qué es posible realizar y qué no.

En el siguiente apartado lo que se incluirá será la descripción específica de lo que se ha podido realizar. Se verá la implementación que se ha desarrollado, y las elecciones tomadas durante el desarrollo. Así mismo estarán incluidas una por una cada parte en la que se subdivide el programa realizado.

El penúltimo apartado será el de pruebas. En él se plantearán las pruebas que comprobarán que la aplicación funciona correctamente y que sirve para su cometido. Se probará que cada módulo del trabajo funcione correctamente y se dirá cuáles han sido los resultados.

Por último, estará el capítulo de conclusiones y líneas de trabajos futuros. Este es un apartado importante que no debe faltar en el proyecto dado que en él se resume lo que se ha aprendido y conseguido con este trabajo, y cuáles podrían ser sus continuaciones. Esto será lo que se incluya en dicho apartado.

Chapter 2: Introduction

• 2.1 Motivation

The growth of technology is something known. It is obvious to say that currently the telephony is extremely widespread, and it is increasingly frequent use of devices such as tablets or SmartTVs

There are currently about 7.2 billion people on earth (of which not all have access to the internet or to existing technologies), and the current number of connected devices is 25 billion. According Cisco[5] predicts, by 2020 there twice devices, ie devices will be 6.58 per person.

improved communication is needed because soon we will see multiple applications with many devices being connected to each other, transmitting information.

In the coming years there will be services that require high levels of bandwidths. Internet of Things is one of the biggest proponents of M2M[6] systems since for running applications, complete independence of people will be needed.

Some of the main objectives in the area of IoT are:

- Smart Grid: transmission measures in the production of services
- monitoring: temperature, light, humidity and regulation
- Emergency services: as alarms, which require a high priority
- eHealth: health care communications devices, require periodicity
- information and navigation: random communications
- other applications that can be elastic, or real-time ...

[5] D. Evans, The internet of things: How the next evolution of the internet is changing everything, CISCO white paper 1.

[6] Machine-to-Machine (M2M) communications: A survey, Journal of Network and Computer Applications
Volume 66, May 2016, Pages 83–105,
http://www.sciencedirect.com/science/article/pii/S1084804516000990

On the one hand there will be applications that require connection-oriented, like TCP protocols. These applications need to safely transmit packets to their destinations successfully receive them, but such connections require resource reservation, which can be many when it comes to applications for many interconnected devices. You can also create congestion by keeping too many connections. In addition, other applications require a less strict communication, either because the information to be transmitted is not as important, or because they need not communicate constantly. This can be done with protocols such as UDP, which send messages without ensuring receipt, without having to be someone necessarily receiving them. It is a less reliable but lighter communication.

EHealth application may need the first type of connection, since a small change to some extent a person of poor health, can be very important to prevent any disease. You can not miss these measures. Instead Crowd application can Sensing that their measures are dispensable and do not mind being lost, so it should be a communication of the second type.

Category	**2014**	**2015**	**2016**	**2020**
Consumer	2,277	3,023	4,024	13,509
General business	632	815	1,092	4,408
Vertical business	898	1,065	1,276	2,880
Total	**3,807**	**4,902**	**6,392**	**20,797**

Figure 3: IoT devices connected sorted by industry [7]

As we can see, there are a lot of future applications, and some may not require resource reservation, but others need this. Regardless of the nature of the applications, you can confirm that you will need a large number of communications, as its most important feature is the intercommunication between a large number of devices.

[7] Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015 http://www.gartner.com/newsroom/id/3165317

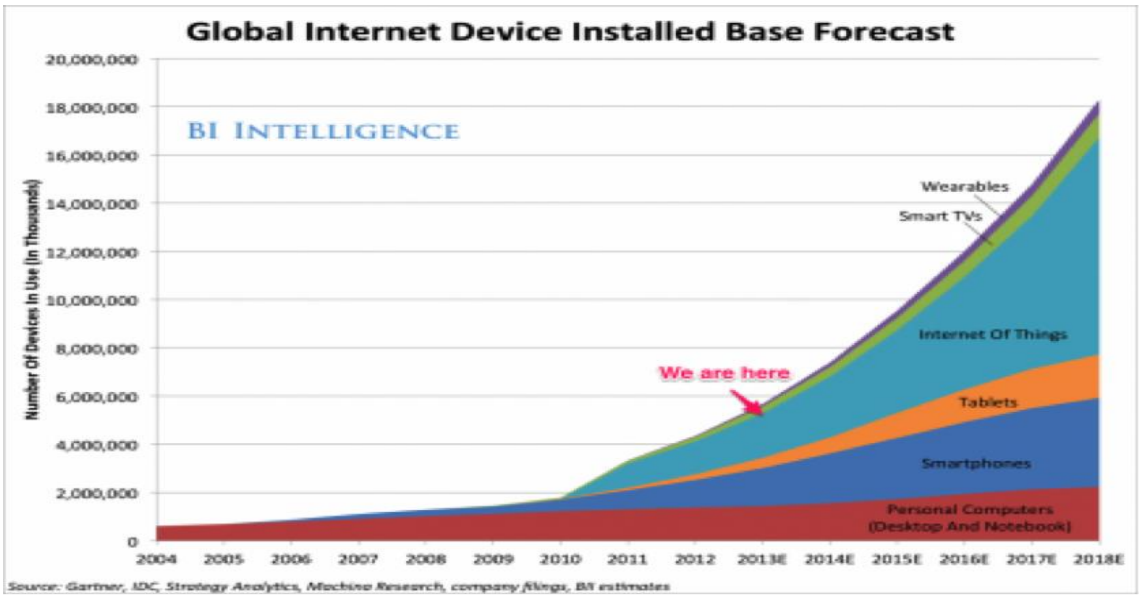

Figure 4: IoT growth relative to other sectors [8]

For all these reasons it is important to develop systems that allow intercommunication between devices that can route messages, duplicate, communicate in different protocols, translate messages from one protocol to another and even store information. It is also important that these elements have properties such as security and scalability.

• 2.2 Objetives:

Because of these needs, this project has decided to design a gateway for M2M technologies, capable of supporting multiple connections, incoming or outgoing, and able to perform the functions listed above. What is desired esque can connect together many elements and groups can be made, so that we can choose for a single issuing multiple receivers or vice versa, allowing for the exchange of messages between different protocols and scalability, running the same program different machines and its location is somewhat transparent when running it.

[8] Gartner ADC, Strategy Analytics, Machine Researching, company filings, BII estimates.

All this will be possible thanks mainly to systems Akka actors and libraries Apache Camel. As discussed in paragraph 2 (state of art), the system of actors Akka is a set of libraries that provides some actors called objects, which represent any type of object and once created, can not be modified. They will be discussed by reference in order to protect the information contained in these actors, and communicate with each other through messages.

Moreover Apache Camel provides a set of producers and consumers, they will be responsible for producing and receiving messages respectively from the outside. Every consumer and every producer will have an IP address and port on which to listen or to send messages. In addition, Camel allows such messages can be sent in different protocols. Camel in components can be found a lot of protocols on which work is permitted. It is very easy with these libraries create a SIP message, or a message of HDFS, without having to develop that field message field.

Thanks to these tools can make the project objective. Both greatly simplify programming, since the design of a Gateway operating in different protocols, that is scalable, secure, asynchronous, allowing clusters is very complicated if created from 0. Fortunately all these options mentioned are already developed and only You must implement what is already created.

Finally, it is noteworthy that the project has not developed the Gateway as a whole, but the aim is to demonstrate that these tools can create such a Gateway in a simple way, you may have these properties, and create a small application that is an example of the capabilities of these tools on a small scale. In paragraph 4 (implementation) will be seen more accurately what the project functions are those that can be implemented and which, even knowing that are possible, have not been developed due to lack of both financial resources and time.

In short, what do you hope to achieve? What is sought in this project is to design a program that can provide evidence that Akka and Camel as tools and java programming language, you can create a simple program that when executed, the machine that is running behave like an element that

multiple devices can access and they can communicate with each other, translate protocols, etc.;

• 2.3 Memory contents

having already seen the motivation that has led to this project, and the goal is, it will happen to detail the sections that will be in the document memory.

First will be the state of the art section in which all the technologies used for the project, which has not been seen throughout the race detail in depth. It is obvious that explain in that paragraph as Java or Eclipse works is useless, it is understood that along engineering have seen these tools and therefore what develops in this section are the technologies that need not be seen as throughout the race, and which are essential for the development of the project.

Following this section, you will see the overall design of the project, with all the elements that would be needed to meet the target, how could perform each part, but making it clear what is possible and what not.

The following section will include what will be the specific description of what has been done. implementation that has developed will be, and the choices made during development. It also will be included one by one each part in the program is divided done.

The penultimate paragraph will be the test. It describes the tests that verify that the application works properly and used to its task will arise. It is tested each module work work properly and tell what were the results.

Finally, it will be the concluding chapter and lines of future work. This is an important section that should be present in the project because it summarizes what has been learned and achieved with this work, and what could be their continuations. This is what is included in that paragraph.

Capítulo 3: Estado del arte

- Herramienta de desarrollo Akka:

  Akka es una herramienta que proporciona un conjunto de librerías que pretenden ayudar a mejorar la programación. La finalidad de Akka es facilitar la escritura de programas, haciéndola una tarea más sencilla.

  Akka proporciona un modelo de actores cuyo objetivo es la abstracción. Éste modelo nos permite diseñar aplicaciones escalables, flexibles, sensibles, tolerantes al fallo y distribuibles. Los actores de Akka son una excelente herramienta para realizar una programación asíncrona, no bloqueante, concurrente y con un modelo de control por mensaje.

  Respecto a la tolerancia a fallos, es posible dado que los actores de Akka son procesos muy ligeros, extendidos a lo largo de muchas máquinas virtuales de java diferentes, con lo que pueden permitir accidentes sin que altere al funcionamiento del resto del programa.

  La localización de la ejecución es transparente, dado que Akka permite ejecutar cada actor en una maquina diferente y que estos se comuniquen totalmente por mensajes asíncronos. Por éste motivo, es una herramienta perfecta para el diseño de aplicaciones distribuidas.

- Actores:

  Como ya se ha explicado, los actores permiten una programación concurrente, distribuida, asíncrona y no bloqueante. Vamos a ver cuáles son las principales características de los actores que nos permiten programar con todas estas ventajas de una forma tan sencilla:

  - Actor por referencia: Los actores son protegidos del exterior mediante el modelo de referencia. Los actores son representados hacia el exterior mediante una referencia, con la cual podremos

realizar todas las operaciones deseadas, manipular al actor que está en otra máquina remotamente, y sobre todo, impedir que se pueda manipular el interior del actor ni controlar su estado.

- Estado: Los actores guardan en alguna variable los estados en que se encuentran los actores. Esto puede explícitamente con una máquina de estados, u otro mecanismo tal como un contador, un conjunto de oyentes o las solicitudes que quedan pendientes. Esta es una información muy importante, pero está muy bien protegida de agentes externos. Además de lo explicado anteriormente, los actores se ejecutan cada uno en su propio hilo, protegiendo su información del resto del sistema. Esto conlleva que no tengamos que preocuparnos por la concurrencia cuando queramos escribir el código de los actores.

  Realmente, Akka ejecutará conjuntos de actores en un conjunto de hilos, pudiendo compartir varios actores un mismo hilo, o incluso que un único actor pueda ocupar varios hilos, pero Akka asegura que este sistema no afecta a que los estados de los actores sean independientes unos de otros.

  Por último, dado que el estado interno del actor es algo fundamental para su funcionamiento, cuando el actor falla y se reinicia, el estado también se creará de nuevo para permitir la recuperación del sistema. Además, opcionalmente, el estado anterior al reinicio puede recuperarse.

- Comportamiento: El comportamiento es la función que debe realizarse cuando el actor procesa un mensaje recibido en un instante de tiempo. Esta función puede cambiar en el tiempo, dado que no querremos que nuestro actor se comporte de la misma forma siempre. Estos cambios de comportamiento se guardarán en variables de estado.

- Mailbox: el objetivo de los actores es el procesamiento de mensajes, mensajes que se envían desde otros actores, de dentro o fuera del sistema. Lo que conecta a los receptores con los emisores es el mailbox de dicho actor, en el que todos los emisores depositan sus mensajes en el momento en que se envían. Hay varias maneras de implementar la cola de un mailbox, pueden estar implementadas como FIFO, o en cambio colocarse por prioridad, o por categorías, pueden estar limitadas a número de mensajes o a cantidad de memoria ocupada, o ser ilimitadas… Otra característica importante de Akka es que el siguiente mensaje de la cola a extraer lo manejará la función de comportamiento actual.

- Hijos: Un actor es potencialmente un supervisor, dado que si dicho actor debe crear un hijo en algún momento de su ejecución para delegar en el alguna sub-tarea, automáticamente las supervisará. La lista de hijos se guarda en el contexto del actor y éste tiene total acceso a ellos. La creación y destrucción de los hijos se realiza de forma asíncrona por lo que no bloqueará a su supervisor.

  Una pieza clave del actor es su estrategia para manejar errores de sus hijos. El manejo de errores es realizado transparentemente por Akka, aplicando una de las estrategias definidas en supervisión y seguimiento para cada error entrante.

- Finalización de un actor: Cuando un actor es terminado, es decir, no termina por un reinicio manejado, sino que es finalizado por sí mismo o por su supervisor, liberará sus recursos, enviando todos los mensajes de su mailbox al deadletters.

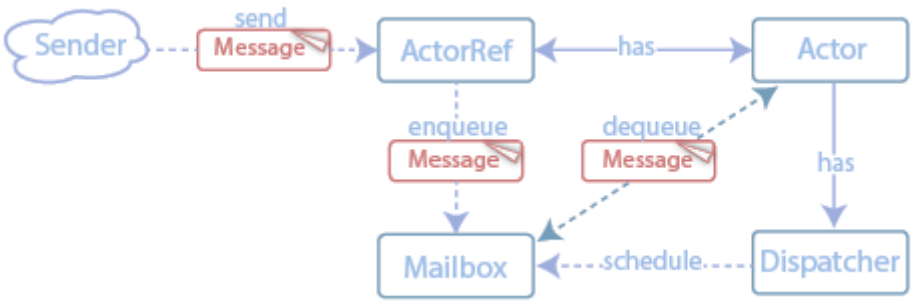

Figura 5: Envío de mensajes entre dos actores

- Apache Camel:

  Apache Camel es un framework de código abierto que permite diseñar reglas de enrutamiento en diferentes protocolos. Incluye APIs para múltiples lenguajes de programación como Java, Spring, XML o Scala.

  Camel sirve para trabajar con cualquier tipo de protocolo de transporte o aplicación tal y como HTTP, ActiveMQ, JMS, JBI, SCA, TCP, UDP, HDFS, SMTP, IMAP, ZeroMQ, SIP, Spark, etc. Permite trabajar con la misma API independientemente del tipo de mensajería que se use. Utiliza URIs para trabajar sobre cualquier protocolo. Así mismo, aparte de especificar una IP y un puerto para dicha URI, también se pueden especificar rutas, para tener diferentes respuestas para el mismo puerto, y diferentes opciones que nos determinarán el comportamiento del programa en la comunicación establecida.

  Camel introduce el sistema de productor-consumidor. El productor será el elemento encargado de generar solicitudes y transmitirlas, mientras que el consumidor será el que las recibe y el que determina que se debe hacer con el mensaje una vez se haya recibido y desencapsulado.

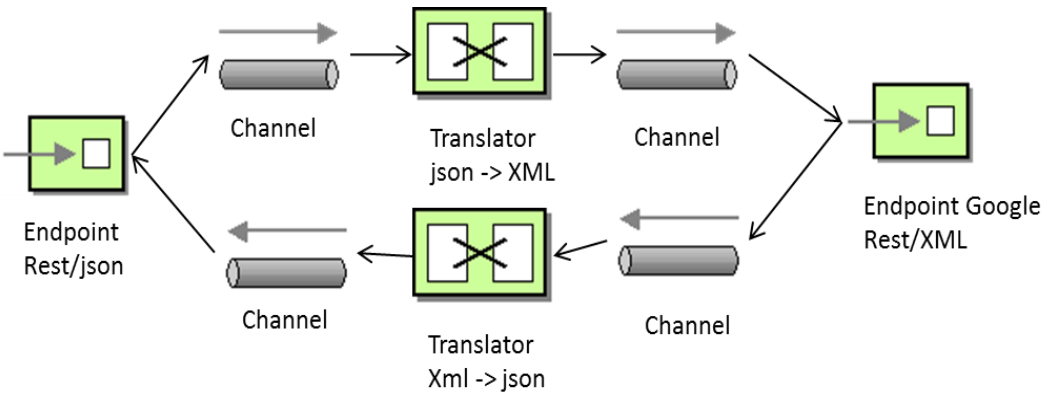

Figura 6: Ejemplo de traducción con Camel

- Apache Maven:

  Maven surge con el fin de simplificar la programación en java en el día a día.

  Es un estándar para construir proyectos, compartiendo librerías o archivos jar, publicar información a través de diferentes proyectos. Está disponible para usarse con cualquier proyecto java.

  Hay varios objetivos que trata de cumplir Maven, tales como hacer que el proceso de desarrollo del proyecto sea fácil, proporcionar un sistema de desarrollo uniforme, proporcionar información del proyecto, o una sencilla modificación de características.

  Proporciona un documento XML llamado POM, en el cual especificaremos las dependencias de nuestro proyecto, con sus respectivas versiones y requisitos. Esto hará que con unas pocas líneas de código XML maven se encargará de descargar de su repositorio las

librerías indicadas. Esto es una clara mejora respecto al sistema tradicional de importación de librerías donde se tenía que descargar cada una, e instalarse en el directorio del proyecto, y cada vez que se ejecutara el programa especificar qué librerías se quieran incluir. Maven modifica todo este proceso por escribir unas sencillas líneas de código en el fichero POM.xml. Además este sistema nos permite ver las versiones de nuestras dependencias, y ver cómo están relacionadas entre sí, para cuando se deseé incluir nuevas librerías o características, saber que versiones se necesitan instalar para que el proyecto sea compatible.

- M2M (machine to machine):

  Cada vez se están haciendo más populares nuevos conceptos referentes a la tecnología y a las telecomunicaciones. En los últimos años se oyen términos como IoT, Crowd-Sensing, Smart-Cities, etc. Todos ellos son proyectos cuya principal característica es de comunicar todos los dispositivos entre sí, para que realicen una determinada función.

  Estas aplicaciones están siendo desarrolladas con unas técnicas diferentes a las habituales hasta el momento en cuando a las comunicaciones, introducen el concepto de M2M, o máquina a máquina. Las comunicaciones habituales han sido siempre iniciadas por personas, ya sea para comunicarse entre sí o ya sea para obtener una determinada información de una máquina si ésta está desarrollada de una forma más inteligente y permite contestar las solicitudes del cliente. Lo que introduce M2M es la comunicación entre máquinas, de forma totalmente autónoma.

  IoT y M2M son dos conceptos que habitualmente se confunden. El primero se refiere al objetivo de la recolección y procesamiento de datos con alguna determinada finalidad por parte de múltiples dispositivos intercomunicados. En cambio, M2M es la propia tecnología que permite la intercomunicación para las funciones deseadas.

  M2M propone proveer una arquitectura para las tecnologías presentes y

futuras, interoperabilidad y confidencialidad y privacidad. Incluye tres dominios principales:

- Dominio de dispositivos: En él se incluyen una gran cantidad de dispositivos que transmiten datos monitorizados a un Gateway.

- Dominio de red: Es la interfaz entre el dominio de aplicación y el de dispositivo. Usa protocolos de largo alcance tanto por cable como inalámbricos.

- Dominio de aplicación: servidores finales y aplicaciones clientes de M2M. Es donde se da lugar a la recolección de datos.

Entre los objetivos de M2M encontramos la transmisión masiva simultanea por parte de muchos dispositivos clientes, fiabilidad (garantía de conectividad), programación de prioridad de mensajes, bajo consumo de energía (se pretende que en la quinta generación los dispositivos puedan durar una década con una sola batería), tráfico imprevisto, baja o nula movilidad, monitorización, seguridad y baja latencia.

Existen múltiples instituciones de estandarización en el M2M: 3GPP, ETSI, oneM2M e IETF.

En cuanto al dominio de dispositivo, se incluye la transmisión desde el dispositivo al Gateway, se debe de tener en cuenta que los dispositivos normalmente son limitados, tienen baterías cargadas, pueden necesitar suspender o pararse para ahorrar energía, puede que no estén siempre conectados, y puede que necesiten comunicarse con dispositivos cercanos.

Las tecnologías más importantes para M2M son WiMax, Wi-Fi, ZigBee, 6LowPAN y Bluetooth, cada una tiene sus ventajas y sus desventajas según cómo se usen. Por ejemplo, ZigBee está diseñado para dispositivos de bajo consumo, con baterías de larga duración, usado para una baja tasa en áreas pequeñas.

Capítulo 4: Diseño

Ya se ha visto en los apartados anteriores, las motivaciones que han llevado a desarrollar el proyecto, el objetivo que se ha propuesto, un resumen de lo que contendrá cada apartado, y el estado del arte, el apartado en el que se habla de las tecnologías más sofisticadas empleadas para el desarrollo del proyecto.

Recopilando lo anterior, se ha dicho que el objetivo es desarrollar un Gateway que permita operar en distintos protocolos, que permita comunicar los diferentes dispositivos conectados, que pueda ser escalable y manejable. Por ello se han propuesto los siguientes módulos en el diseño:

- Interfaz: Este módulo tratará sobre como el usuario podrá manejar la aplicación para crear, modificar, o eliminar actores, crear agrupaciones, etc.

- Actores: En este apartado se verán las características de los actores y cómo se diseñarán.

- Protocolos: Se explicarán las opciones más importantes en cuanto a protocolos de enlace, transporte y aplicación.

- Estructura distribuida: Se explicará cómo se realizará una estructura distribuida y cómo se utilizaría para la aplicación.

- Ejecución: Aquí se detallará el funcionamiento del programa una vez sea arrancado

¿Por qué Java? En realidad, cualquier lenguaje de programación podría realizar funciones de comunicación con dispositivos externos mediante el uso de sockets y librerías que permitan el envío y recepción de mensajes. Pero Java es un lenguaje intuitivo, que gracias a la orientación a objetos se facilita mucho su uso, y lo más importante, hay una gran cantidad de

APIs que permiten reciclar código ya escrito sin necesitar saber cómo está hecho.

4.1 Interfaz gráfica

Ahora se verá la interfaz que se desea tener para la aplicación: el objetivo es que cuando se arranque el programa aparezca esta interfaz, desde la cual se pueda programar todos los actores que se deseen tener. Lo primero que habrá en la interfaz será una vista en la que aparezcan los actores existentes, inicialmente estará vacía.

En algún lado de la ventana, estarán los botones que nos permitirán realizar acciones. Uno de ellos será el botón de añadir, con el cual se abrirá una ventana emergente donde se elegirá el protocolo, dirección IP y puerto en el que querremos escuchar. Lo configuraremos, y se creará el actor. Se volverá a la vista inicial donde aparecerán los actores creados. Si se selecciona uno o varios actores en esta vista, se podrán eliminar con un botón que habrá para ello.

Por otra parte se deberán incluir los botones de crear agrupaciones y de distribuir los actores en diferentes máquinas. En la primera opción se crearán grupos, en los cuales se determinarán los actores que producen mensajes y los actores que reciben dichos mensajes.

La estructura distribuida se programará determinando qué actores se ejecutarán para cada IP, de tal forma que cuando el programa se ejecute en diferentes máquinas con diferentes IPs, cada actor se creará en una máquina distinta.

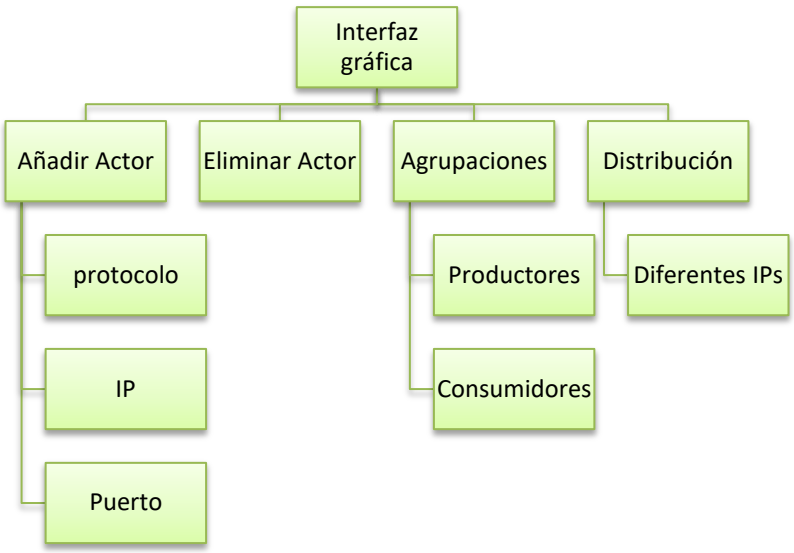

Figura 7: Diagrama de las funciones de la interfaz

• 4.2 Actores

Este es el módulo central del diseño. Como ya se ha venido advirtiendo, los actores son la pieza que permite realizar comunicaciones entre diferentes dispositivos conectados.

Hay dos tipos de actores, los consumidores y los productores. Los primeros serán los que reciban mensajes y los envíen a los segundos. Los productores serán los encargados de recibir mensajes de consumidores y enviarlos al elemento al que estén enlazados.

Los consumidores cuando reciban un mensaje, comprobarán que tiene el formato correcto, mostrarán por consola la información más importante de la cabecera y a continuación lo mandarán al productor.

Todos los actores, productores y consumidores, funcionan en un objeto de Akka llamado Actor System, el cual agrupa los actores de nuestra aplicación. Se pueden crear tantos sistemas de actores como queramos y que cada uno tenga su propio grupo de actores pero en este caso solamente interesará uno, ya que para las agrupaciones se usará otra herramienta de Akka llamada Event Bus.

Event Bus es una herramienta con la cual se pueden realizar agrupaciones del tipo suscriptor-publicador de múltiples de formas. Entre estos tipos se encuentran modelos como actores que se suscriben a un *"topic"* y cuando algún otro actor publica algo en este *"topic"* se le enviará a todos los demás. Hay otros tipos similares pero hay uno que en especial es el que interesa, y es en el que los *"suscribers"* quieren recibir mensajes de un determinado actor asique se suscriben a él. En este modelo no hay que suscribirse a un tema sino a un actor. Todos estos enlaces se guardan en nuestro bus, que previamente hemos creado con una plantilla que Akka nos proporciona en la documentación. Cuando un actor recibe un mensaje, lo publica en el bus y todos los actores suscritos a ese actor lo reciben. En nuestro modelo, los suscriptores serán los productores, y los publicadores serán los consumidores.

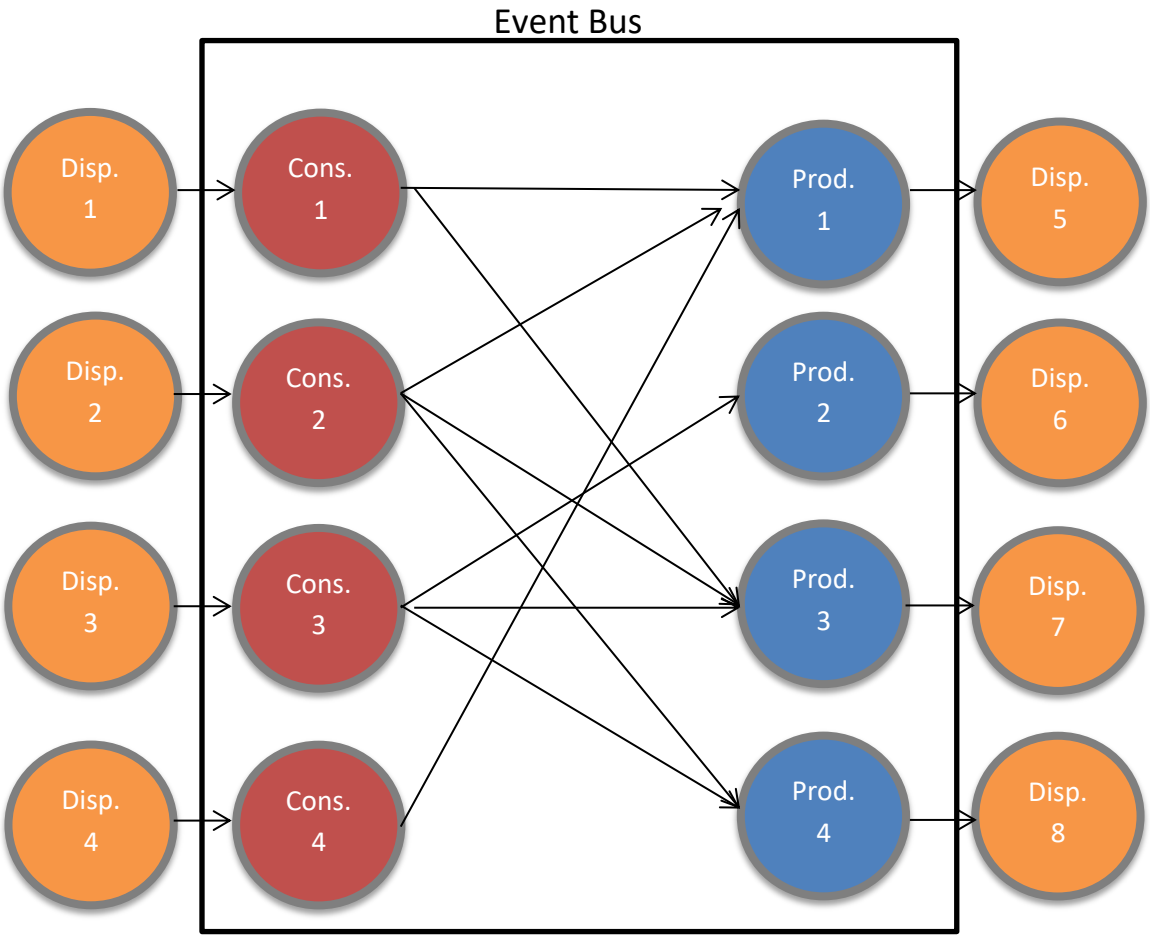

Figura 8: Relación de actores con Event Bus

Como muestra el esquema, en el bus se guardan los enlaces entre actores, los cuales estarán conectados a los dispositivos que pueden ser de cualquier tipo, ya sean sensores, o actuadores, o cualquier otro tipo de elemento que vaya a ser usado en nuestra aplicación. No es necesario que haya un actor para cada dispositivo, eso sería lo mejor para darle un trato diferenciado a cada elemento, pero pueden conectarse varios a un mismo actor y tratarse todos de la misma forma.

Por ejemplo si unos sensores detectan que en una calle está habiendo mucho tráfico, y se quieren poner los semáforos del próximo cruce a verde se pueden agrupar los cuatro semáforos para el mismo actor, y así cuando se mande la información del tráfico, los cuatro se alternen a la vez dejando paso a la calle más congestionada. Esto es sólo un ejemplo de cómo puede interesarnos unir diferentes dispositivos a un mismo actor, pero como se enunció en el apartado de objetivos, nuestro diseño se reduce a un Gateway orientado a IoT que opere en diferentes protocolos y que permita interconexiones, no a una aplicación concreta.

Respecto a los protocolos que se usarán en el diseño, serán los más importantes, como TCP, UDP, HTTP, HDFS, SMTP, etc, aunque realmente se pueden implementar una gran cantidad de protocolos, gracias a la plataforma Apache Camel, antes mencionada en el apartado 2 (estado del arte), que nos permite de una forma muy sencilla e intuitiva mandar mensajes en una gran cantidad de protocolos. La manera de implementar un protocolo u otro es muy sencilla y solo se diferencia en la URI y las librerías empleadas, asique para el inicio de la aplicación no se tienen que implementar una gran variedad de protocolos, sino que eso puede fácilmente ir desarrollándose según se de uso al programa.

• 4.3 Protocolos

Debido a la gran cantidad de dispositivos, y la variedad de protocolos que emplean, hay que plantear diferentes tipos de comunicaciones a la hora de realizar el programa.

Por una parte están los protocolos de nivel de enlace. Entre estos hay mucha variedad y cada uno tiene sus ventajas y sus desventajas: algunos vienen mejor para conexiones cercanas como puede ser Ethernet, y otros, especialmente los protocolos inalámbricos, están pensados para dispositivos que estén alejados de sus puntos de acceso a la red.

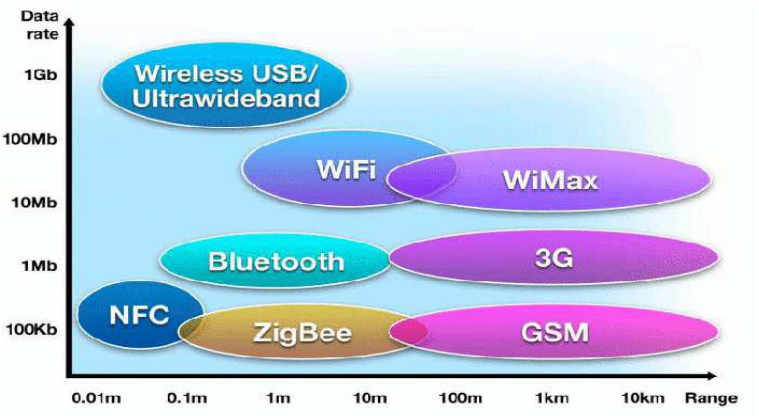

Figura 9:Tabla de protocolos inalámbricos según su alcance y tasa de datos

Es importante que el diseño del Gateway permita operar en varios de estos protocolos. Los más interesantes para el Gateway de este proyecto serían al menos WiMax o LTE, Ethernet, SDH y ZigBee.

WiMax y LTE, son los protocolos que se usarán para los dispositivos móviles conectados inalámbricamente. Son preferibles a 3G y GSM porque tienen mucha mejor tasa de envío de datos. De entre estos dos protocolos, WiMax ha sido muy importante en Estados Unidos, pero en Europa lo que ha destacado es LTE, que también está desarrollada en EEUU asique de entre estos dos protocolos, el escogido será LTE porque es al que se dirige la tecnología actual de 4G. El único inconveniente de este protocolo es que aún no está totalmente instalado en algunos lugares. Si la aplicación para la que se usara el Gateway tuviera muchos dispositivos en zonas rurales, como por ejemplo para una aplicación relacionada con la climatología, habría que comprobar si en dichas zonas hubiera buena cobertura para estos protocolos y en caso de no ser así, cambiarlos por protocolos de 3G como UMTS, que está más extendido.

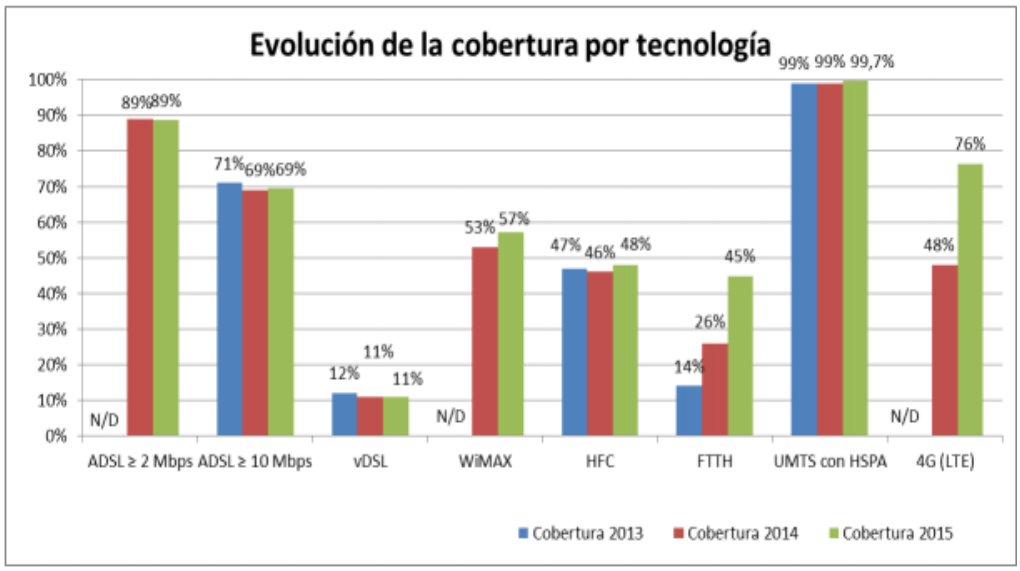

Figura 10: Evolución de la cobertura nacional por tecnología[9]

[9] Informe de cobertura de banda ancha en España en el primer trimestre de 2015 http://www.minetur.gob.es/telecomunicaciones/banda-ancha/cobertura/Documents/cobertura-BA-1trimestre2015.pdf

En cuanto a los otros protocolos, Ethernet y SDH son fundamentales para las conexiones por cable, tanto de cobre como de fibra óptica. Por último, ZigBee compite con Bluetooth por ser el protocolo inalámbrico a corta distancia. Mientras que Bluetooth es usado principalmente para la transferencia de ficheros, ZigBee se usa para comunicaciones de pequeños mensajes. Está pensado para mallas de dispositivos que envíen mensajes eventualmente y con la característica del bajo consumo energético. Es por eso que ZigBee es el protocolo que mejor encaja con el proyecto.

En cuando a protocolos de nivel de transporte, TCP y UDP son totalmente necesarios. Para aplicaciones a tiempo real, como la transmisión de mensajes de los dispositivos, UDP es el adecuado, no requiere tener conexiones establecidas y consume menos ancho de banda. Es el que se usará para dispositivos cuyos mensajes no sean imprescindibles, dado que frente a congestiones y pérdida de paquetes, UDP no garantiza el reenvío ni que lleguen al destino. En cambio TCP si lo garantiza, a cambio de un mayor consumo de ancho de banda, pues en TCP se controla el flujo de mensajes. Este será el protocolo que se usará para dispositivos cuyos mensajes sean más importantes, y no puedan perderse. Este puede ser el caso de sensores de vigilancia, sensores de cercanía en vehículos, etc.

En cuanto al último nivel de protocolos, los de aplicación, gracias a Camel hay una gran cantidad de posibilidades y todas fácilmente programables. A continuación se mencionan algunos de los protocolos que se pueden desarrollar y sus características. HTTP es muy útil gracias a su funcionamiento. HTTP se comunica mediante mensajes textuales, es un protocolo muy extendido y no necesita guardar estado. SiP funciona de una manera similar y es muy útil para el control de transferencias multimedia, como sería el caso de dispositivos como micrófonos o videocámaras de vigilancia. HDFS es muy útil para sistemas de almacenamiento de big data. En el siguiente apartado se verá qué utilidad tendría para el proyecto. También es muy útil para el Gateway que implementara un protocolo de correo, como IMAP, POP3 o SMTP. Si hubiera algo importante emitido por algún sensor de los conectados, sería muy útil que el Gateway hiciera una recopilación de lo que ha ocurrido y lo enviará por correo al cliente de la aplicación.

• 4.4 Estructura distribuida

Otro de los módulos del diseño es el de la distribución del programa en diferentes computadoras. El objetivo de esto es que se pueda escalar la aplicación para poder tener la cantidad que queramos de actores. De no hacerlo así, lo que puede pasar es que si se quisiera usar el programa para una red de dispositivos más grande de lo que el ordenador/servidor puede procesar se bloqueé o tenga que tirar mensajes.

Con una distribución de la aplicación en diferentes servidores se pueden evitar estos problemas de congestión. Hay proveedores de servicios de distribución que no requieren de una determinada capacidad de almacenamiento, de procesamiento, o de un número de servidores, sino que se puede contratar el servicio y una determinada capacidad de procesamiento y de memoria. Cuando parezca que la aplicación va a requerir más capacidad, ya sea de memoria porque se empiece a almacenar más información de los dispositivos, o de procesamiento, porque haya más dispositivos y por tanto más actores, podemos ampliar las capacidades contratadas.

Respecto a cómo programar la distribución, Akka ofrece la posibilidad de configurarlo. Akka permite que los actores que creemos sólo se ejecuten en una determinada máquina. Esto viene determinado por la IP, se puede definir a cada actor una IP, de tal forma que solamente se crean los hilos del actor cuando el programa se esté ejecutando en la máquina que tenga esa IP.

Ya se han planteado las dos partes de la distribución, la del hardware, que se puede realizar o bien contratando a un proveedor de servidores, o comprándolos, y la del software, que se puede hacer fácilmente gracias a Akka.

A parte de esta funcionalidad que se acaba de desarrollar acerca de ejecutar el mismo programa en diferentes máquinas y así compartir recursos, se puede realizar otra opción, y es que solamente compartan

recursos de procesamiento y que el alojamiento de los datos sea remoto, es decir, cuando se reciban mensajes, los servidores contratados los recibirán, traducirán y enviarán; y cuando se necesite almacenar la información recibida, se enviará a otros servidores cuya función será el almacenamiento.

Para esta funcionalidad es muy útil Camel, ya que con esta herramienta se pueden enviar mensajes en HDFS, que es un protocolo utilizado para la transferencia de archivos o ficheros a estructuras de ordenadores descentralizadas. Esto lo hace gracias a un sistema de Datanodes y Namenodes. El conjunto de servidores donde se almacenarán los archivos deben de tener instalado HDFS. Uno de los servidores será el Namenode, que es al que se conecta nuestro Gateway para enviar los archivos. Éste, recibe los mensajes HDFS, recupera los archivos, los divide en partes y cada parte la envía al conjunto de servidores, los Datanodes. Un archivo no tiene por qué estar localizado sólo en una máquina, cada parte puede estar situada en un servidor distinto, y cuando se quiera recuperar dicho archivo, el Namenode guarda la localización de cada parte, asique se la pide a los Namenodes y vuelve a construir el archivo.

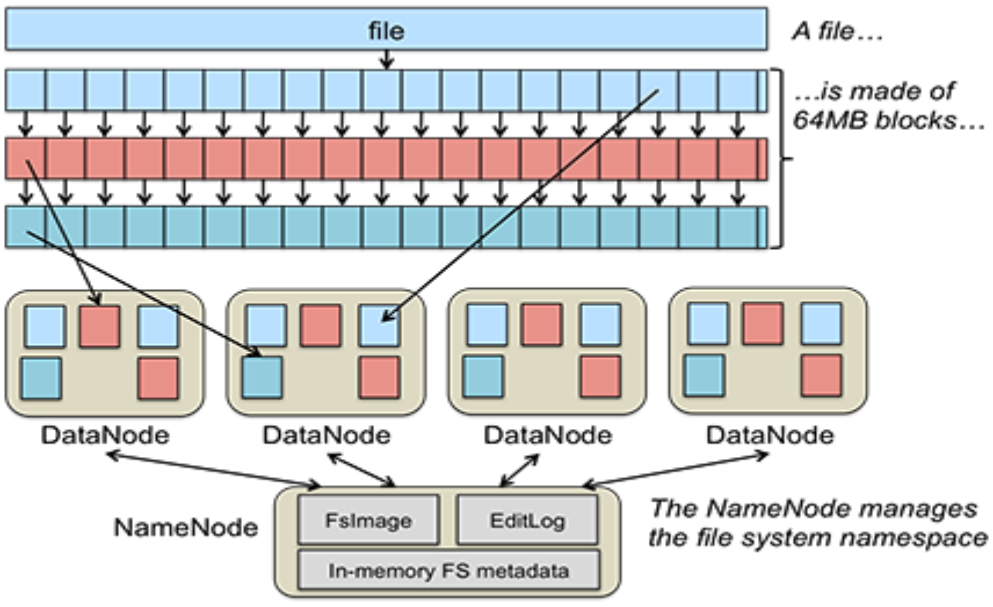

Figura 11: Representación del funcionamiento de HDFS

• 4.5 Ejecución

Lo que hará el programa al arrancarse es inicialmente mostrar la interfaz gráfica, donde veremos los actores creados. Inicialmente estará vacía la lista, y como se mencionó en el apartado 3.1, pero se pueden ir creando los actores con los protocolos, IPs y puertos que se deseen, además de determinarse en qué máquina se ejecutarán. Cuando estén creados los actores deseados, se empezarán a crear las agrupaciones de actores para que puedan comunicarse entre sí con el event bus.

Una vez realizado esto, ya la aplicación puede funcionar, habrá que conectar los dispositivos a los servidores, y viceversa, con su IP correspondiente y puertos en los que están escuchando los actores. Entonces ya podrá funcionar correctamente todo.

Capítulo 5: Implementación

En este capítulo se va a explicar cómo ha sido el desarrollo real del proyecto, cómo se ha programado, y qué partes del diseño son las que se han podido realizar y qué partes no, ya sea por cuestiones económicas, falta de conocimientos, o falta de tiempo dado que la extensión del proyecto se ve limitada por otros factores académicos.

Las partes del diseño en las que se centrará la implementación es en la de los actores, los protocolos, y por supuesto también la ejecución.

Como breve resumen de lo que habrá explicado a continuación, se hablará de los actores, de cómo funciona el event bus, y de cómo se ha planteado la ejecución para poder probar que este sistema funciona. Los actores, se dividen en dos tipos, los consumidores y los productores, unos reciben mensajes y otros los envían. Por otra parte también se hablará de otro tipo de actores que no son de ninguno de los tipos anteriores, que servirán para determinar el comportamiento de los productores al recibir mensajes, puesto que éstos cuando reciben un mensaje lo reenvían sin elección de qué hacer con él. Ambos tipos de actores, consumidores y productores, están comunicados gracias al event bus, herramienta que proporciona Akka para poder enviarse un mensaje a múltiples destinos según qué criterios. El de este diseño es el modelo de suscriptor-publicador. En cuanto a la ejecución, en resumen, es crear varios actores en diferentes máquinas, comunicándose unos con otros, y generar algunos mensajes.

• 5.1 Actores

Como ya se dijo en el apartado 3 (diseño)**,** éste es el módulo más importante del diseño. En él vienen incluidos los actores productores y consumidores de cada protocolo, y la parte del event bus que usamos para que pueda haber múltiples receptores de un mensaje enviado.

Empezamos viendo cómo funcionan los actores:

- Productores:

  Este será el primer tipo de actor que veremos ya que es el más sencillo. En primer lugar, los productores tienen un atributo que será la uri con la cual Camel reconoce el protocolo que queremos desarrollar, la IP y puerto que vayamos a configurar. También a la uri se le pueden añadir opciones[10] a continuación del puerto. Toda la configuración de sockets y construcción de mensajes es transparente a la programación gracias a Camel.

  El constructor, con el cuál se crean los objetos de esta clase, tendrá dos argumentos, la ip y el puerto, con los cuales se inicializa la uri.

  Por último, todo productor tiene un método heredado llamado getEndpointUri, que sirve para que se configure una conexión con la dirección dada, de tal forma que cuando reciba cualquier mensaje el productor, lo enviará a esa dirección.

  En la clase del productor, no se puede elegir qué hacer con el mensaje que se recibe, cuando le llega, lo envía a la dirección vinculada, asique para poder manipular el mensaje, o incluso para modificar el objeto, conviene que haya un actor intermediario entre el consumidor y el productor. Este actor lo explicaremos más adelante en el apartado notCamel.

  Por último, a continuación se enseña el código de un productor de tipo HTTP.

[10] Apache Camel documentation: How do i configure endoints? http://camel.apache.org/how-do-i-configure-endpoints.html

- Consumidores:

  Estos son los Actores que más desarrollo tienen. Al igual que productores, tienen un atributo que es la uri a la que están vinculados. La función de un consumidor es la de recibir mensajes y reenviarlos a los productores, asique la uri debe pertenecer a la máquina en la que están alojados, es decir, debe contener la IP de la máquina. Por otra parte, tienen un segundo atributo, el bus al que escribirán los mensajes.

  El constructor de la clase es similar al de los productores, tiene como argumentos la IP y el puerto igual que en la clase anteriormente desarrollada, pero los consumidores además tiene un tercer argumento, el bus, que inicializará el atributo del mismo tipo.

  A continuación hay un método getEndpointUri que tiene la misma función que en los productores, utilizar el atributo uri para vincular ese objeto a una ip y puerto donde aceptará conexiones del protocolo establecido.

  El siguiente método es el onReceive, heredado de la clase UntypedConsumerActor. Es el método que determina que se hace cuando el objeto recibe un mensaje, por eso tiene un argumento del tipo Object. En el método, primero se comprueba de qué tipo es el objeto recibido, si es un objeto de tipo mensaje de Camel, se imprime por pantalla cuál ha sido el actor que ha enviado el mensaje. A continuación se separa la cabecera del cuerpo, y con un bucle se van imprimiendo todos los campos de la cabecera para dar información sobre el mensaje. Cuando se han impreso los campos, se envía el cuerpo del mensaje convertido a String y encapsulado en un tipo de objeto llamado notification al bus previamente inicializado al crear el objeto. El objeto notification es con el que funciona el bus, más adelante se explicará su clase y como el bus lo utiliza.

Si en lugar de ser un objeto de tipo mensaje de Camel, es de tipo String, cadena de caracteres, se envía directamente al bus.

Por último, hay otro método que es un setBus, por si se quiere modificar el bus. En realidad no se usa este método en el programa creado, pero podría usarse desde el método onReceive. Dado que se trabaja con referencias a actores y no con los objetos en sí, desde el archivo donde se ejecuta el main no se puede acceder a los métodos de un objeto con el que trabajamos por referencias, la única forma de poder llamar a un método de esa clase es desde dentro de ella, y la única manera de acceder a al interior de la clase desde fuera es enviando mensajes al actor referenciado.

Dado que solamente se puede acceder al actor mediante mensajes, éstos tienen que tener el formato que se haya establecido. En nuestro caso, el método onReceive solamente admite mensajes Camel y cadenas de caracteres y las reenvía pero si por ejemplo se quisiera usar otra opción como modificar el bus, se tendría que enviar un mensaje al actor y el cuerpo sería el objeto bus. Desde el método onReceive se comprobaría si es un objeto de tipo bus y si es así se llamaría a la función setBus para modificarlo. En resumen, para llamar al método setBus no basta con mandar por mensaje un bus al actor, sino que se debe de contemplar la posibilidad de recibir ese tipo de objeto por mensaje y llamar a ese método.

Como ya se mencionó en el apartado 2 (estado del arte), el usar referencias a actores en lugar de los objetos en sí da una ventaja y es que al funcionar por mensajes en vez de por métodos, los mensajes tienen que tener el formato previamente establecido o si no, el método onReceive lo descartará. Hace que sea más difícil acceder a los métodos de un objeto desde fuera de él.

- Actores notCamel:

  Éste es el último tipo de actor creado. No cumple ninguna función importante pero sin él, los productores enviarían todo tipo de mensajes sin controlar que tengan el formato correcto, o mejor dicho, que no sean objetos de la clase que se deseen. Por lo general solo se querrá trabajar con objetos de tipo String o CamelMessage. A continuación se va a explicar cuál es el funcionamiento de los actores NotCamel.

  Lo primero que se ve en un actor notCamel son sus dos atributos, como en cualquier otro actor. En éste, son un id y un actorRef, que será el productor al que se desea enlazar.

  A continuación hay un constructor, como se ha visto hasta ahora, con el que se inicializan los dos anteriores atributos.

  Ahora se verá su verdadera utilidad al explicar el método onReceive. Los actores productores carecen de este método, asique con un actor normal sin ser de tipo camel podremos simular el método onReceive de un productor.

  Lo que se hará será comprobar si el objeto recibido es de tipo notificación, que como ya se ha comentado, es la clase de objeto con la que trabaja el bus. En caso de ser de tipo string, se supone que es un mensaje de prueba o similar, asique se imprimirá por consola. En caso de ser un mensaje de tipo referencia a actor, se llamará a la función setRef con la que se cambiará el atributo actorRef, es decir, se modificará el productor al que está vinculado. Esto está desarrollado así por si se ha cambiado la configuración de algún dispositivo, se cambia su ip o puerto, o el protocolo con el que se comunica, el productor que envía el mensaje no puede modificarse, solo se puede crear un nuevo productor con la nueva configuración y reemplazarlo por el anterior. Por último, si lo que recibe el actor notCamel es una notificación, se la envía al productor y este automáticamente lo envía a la ip y puerto que tenga en su configuración.

- Event Bus:

  Esta es el último apartado del módulo de actores, y como ya se ha mencionado anteriormente, es el que da la posibilidad de escribir desde un actor a varios.

  Sin el event bus, para poder escribir desde un actor a otro, en el método onReceive hay que usar el método tell sobre el actor al que se deseé escribir. Con event bus, basta con inicializar un objeto bus en la ejecución, mandarlo como argumento a la hora de inicializar los actores que envían mensajes, y en el onReceive escribir la orden publish.

  Para poder realizar esta función, hay que crear dos clases, una es la clase notification de la que ya hemos hablado anteriormente. Los objetos de esta clase serán los mensajes que enviaremos de un actor a otro a través del bus, habrá tres atributos, uno es el String que será el mensaje en sí, es el cuerpo de los camelMessage que reciben los consumidores y lo envían a los productores. El segundo atributo es un id, para poder organizar las notificaciones si se quisieran almacenar, ordenarlas, eliminarlas, etc. Por último, el tercer atributo. Este atributo es un objeto de tipo ArtorRef, que será una referencia del actor que creó el mensaje, así, los productores que reciban la notificación, sabrán que consumidor fue el que la creó.

  Por otra parte, hay que crear una clase que en nuestro caso se llama ActorBusImpl. El código de esta clase viene dado por Akka en su documentación[11] y tan solo consiste en una clase que hereda de una ya establecida por akka en su librería akka.event.japi.ManagedActorEventBus, y sobrescribir dos métodos de su superclase. Esta clase que acabamos de mencionar será la que se cree y en la cual se almacenan los enlaces entre los actores.

[11] Akka Event Bus: http://doc.akka.io/docs/akka/snapshot/java/event-bus.html

El funcionamiento del bus es el clásico de suscriptor-publicador. El publicador en nuestro caso es el consumidor, el cual publica en el bus notificaciones, y automáticamente los suscriptores la reciben. Los suscriptores serán los actores notCamel, manejarán la notificación con su método onReceive, y lo reenviarán a su productor asignado.

5.2 Protocolos

Como se mencionó en el apartado 3.3 (protocolos), es importante definir qué protocolos se van a desarrollar. Es importante que el Gateway que se desarrolle tenga posibilidades de comunicarse de múltiples formas para poder conectarse a los dispositivos que hagan falta para la aplicación que se le dé al Gateway.

Sin embargo, debido a la falta de recursos, hay que discriminar entre ciertos protocolos. En el diseño se planteó que debería haber protocolos de nivel de enlace, de nivel de transporte y de nivel de aplicación. Los dos últimos se pueden implementar con software, pero en cambio, el primer nivel de protocolos necesita hardware para ser implementado, por tanto hay que descartar estos protocolos a la hora de implementar el proyecto, dado que será únicamente un software, concretamente un programa en java, sin una estructura física con interfaces que permitan los protocolos del nivel de enlace.

Bien, una vez incidido en que solamente se emplearán protocolos de los niveles de transporte y aplicación, se especificarán cuáles serán los protocolos que se empleen: Para la implementación los protocolos utilizados serán TCP, UDP y HTTP.

En el apartado 3.3 se hico una explicación de por qué se necesitan implementar estos protocolos. Podrían haber sido más, pero el proyecto realizado tan sólo pretende comprobar que con las tecnologías empleadas (Akka y Camel), se pueda realizar un Gateway. Para poder comprobarlo basta con elegir varios tipos de protocolos y probar a hacer

actores con ellos, e intentar comunicarlos entre sí. No es importante que se desarrollen actores de todos los protocolos mencionados en el apartado 3.3, sino elegir algunos y comprobar que funcione.

Se eligieron estos tres protocolos, TCP, UDP y HTTP porque son los más intuitivos. Pueden funcionar punta a punta sin intervención de servidores intermedios, y sin configuraciones muy complejas. Además, la implementación de cada uno de ellos es exactamente la misma, tan solo cambia la manera en la que se configura la URI

5.3 Ejecución

Dado que no se dispone de medios suficientes como para probar el programa con dispositivos que hagan la vez de puntos finales de una aplicación, lo que se hará será probar a conectar entre sí los actores de dos máquinas distintas.

Esta ejecución debería de ser suficiente para poder comprobar que nuestro programa funciona, que se puede conectar nuestra máquina con otra y transmitirse mensajes. En cuanto se pruebe que dos ordenadores se pueden comunicar entre sí con este programa, se puede concluir que un ordenador por si solo podría recibir y enviar mensajes de los dispositivos conectados, siempre y cuando se programen primero los actores suficientes como para admitir las conexiones necesarias.

En el apartado siguiente (Pruebas), se hará una comprobación de que estos dos ordenadores se comunican correctamente con el programa desarrollado.

Capítulo 6: Pruebas

Lo que se hará en este apartado será, como ya se ha mencionado anteriormente, conectar dos ordenadores con el programa realizado y ver que funciona correctamente la comunicación.

Para ello, lo que se hará será crear varios actores, de diferentes protocolos y usando el event bus para duplicar mensajes, y ver si el resultado es el esperado.

Para esta prueba, los actores serán:

- PC 1: un productor HTTP
- PC 2: un consumidor HTTP
- PC 2: dos productores de UDP conectados al consumidor anterior
- PC 1: dos consumidores UDP
- PC 1: un productor TCP unido a uno de los dos consumidores anteriores, y tres productores, uno de cada protocolo, unido al otro consumidor anterior.
- PC 2: dos consumidores de tipo TCP, uno de UDP y otro de HTTP para conectarse a los productores anteriores.

Por tanto, los mensajes intercambiados serán los siguientes:

- Un mensaje HTTP (PC1 → PC2)
- Dos mensajes UDP (PC2 → PC1)
- Un mensaje TCP (PC1 → PC2)
- Tres mensajes, uno de cada protocolo (PC1 → PC2)

Por último, se enumerará lo que deberá recibir cada ordenador, es decir, lo que debería de mostrar por pantalla cada ordenador:

- PC1:
    - Productor HTTP envía mensaje
    - Dos consumidores UDP reciben mensaje
    - Un productor TCP envía mensaje

    - Un productor de cada tipo (3 en total) envía mensaje
- PC2:
    - Consumidor HTTP recibe mensaje
    - Dos productores UDP envían mensaje
    - Un consumidor TCP recibe mensaje
    - Un consumidor de cada tipo (3 en total) recibe mensaje

Bien, una vez enunciado todo esto, lo que se espera ver en los resultados de la prueba, veremos si es lo correcto al ejecutar la aplicación en ambos ordenadores, y en uno de ellos enviando el mensaje “Prueba”.

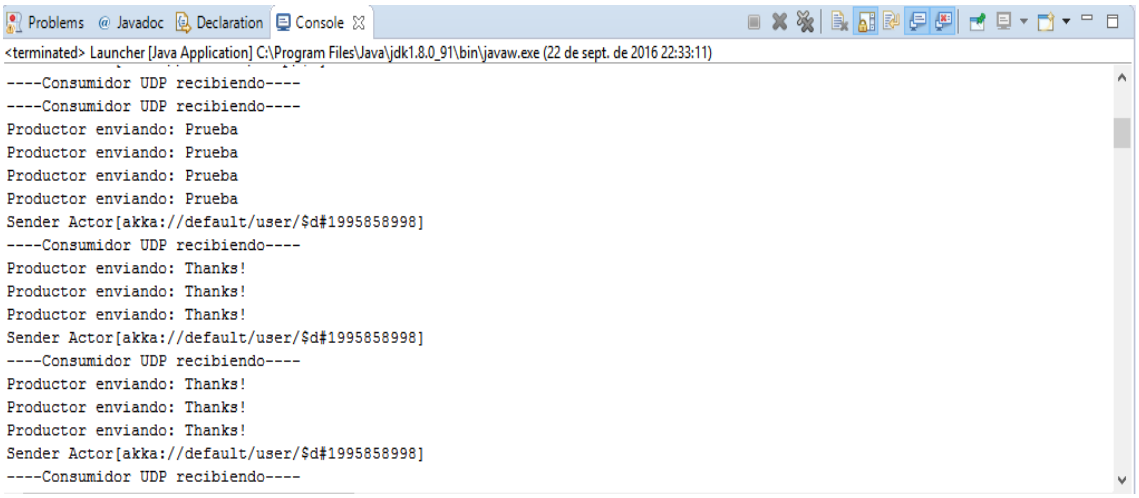

Figura 12: Impresión en consola del PC1

Vamos a analizar la captura anterior: Lo que se muestra en ella es solo una pequeña parte de los mensajes impresos en la consola, hay que tener en cuenta que lo que hacen los actores es establecer una conexión, no solamente enviar mensajes, por lo que se estarán constantemente enviando los mensajes que mantienen la conexión establecida.

También hay que mencionar que solo los consumidores son los que pueden imprimir el tipo de mensaje que se ha enviado, dado que los productores según reciben el mensaje lo envían, sin posibilidad de manipularlo. Como se dijo en el apartado 4.1 (actores), para poder manipular el mensaje de los productores, existen un tipo de actores denominados actores notCamel, que harán el método onReceive en lugar de los productores. El problema es que los actores notCamel no pueden distinguir el protocolo del consumidor que le envió el mensaje, ni tampoco el protocolo del productor al que se lo va a enviar. El protocolo del consumidor se podría ver si dentro del mensaje de caracteres que se envía, se

especificara el protocolo, pero lo importante sería que supiera qué protocolo sigue el productor relacionado con dicho actor notCamel, y esto no se puede hacer dado que el actor notCamel conoce a su productor por una referencia a actor, un objeto de tipo ActorRef, pero no conoce al objeto en sí que se encargará de administrar el mensaje que envía a esa referencia, por tanto no conoce tampoco qué tipo de instancia es.

Bien, toda esta aclaración lo que viene a decir es que el impreso en consola que hacen los actores notCamel al enviar mensajes es el mensaje que dice "productor enviando: …", en nuestro caso, al ser el mensaje de prueba "Prueba", la línea impresa que corresponde a los productores es la de "Productor enviando: Prueba".

El primer mensaje enviado, el que va desde el primer productor http en PC1 hasta el consumidor http en PC2, no aparece en la consola porque el mensaje se envía directamente desde el productor, sin pasar por el actor notCamel, asique no se imprime el mensaje. Pero salvo este, si se puede reconocer los demás mensajes: Los dos primeros mensajes son los de dos consumidores UDP que están recibiendo, como vimos anteriormente, PC2 iba a enviar a PC1 dos mensajes UDP, asique por ahora todo va bien.

Lo siguiente que se ve es cuatro líneas de productores enviando mensajes, estos son el de TCP enlazado a uno de los consumidores UDP, y los otros tres son uno de cada protocolo, enlazados al otro consumidor UDP. No se puede distinguir a qué actor pertenece cada mensaje de productor, pero más adelante en la captura de tráfico se podrá ver con más claridad cada mensaje.

A continuación se verá la captura de pantalla del otro ordenador:

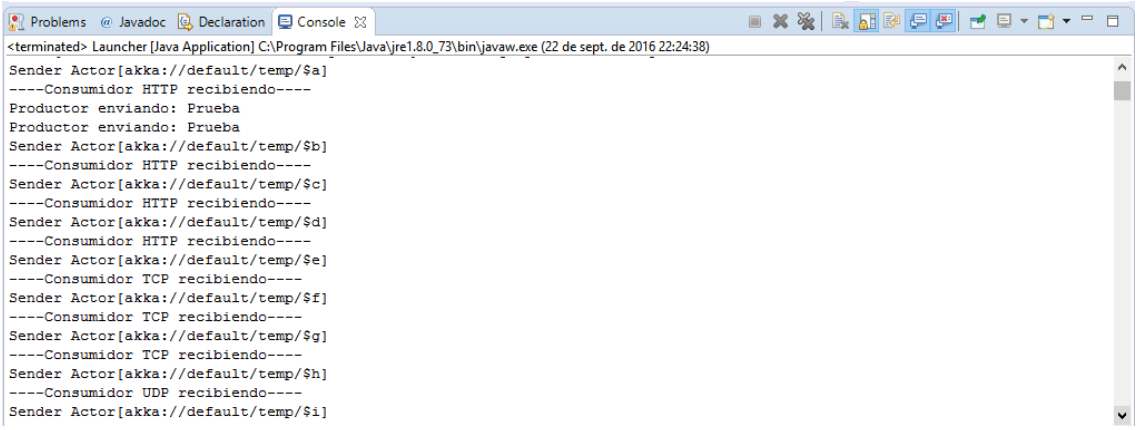

Figura 13: Impresión en consola del PC2

En esta captura se puede ver claramente como nada más empezar el consumidor HTTP ha recibido un mensaje, este es el que en la captura anterior no se veía, pero en esta si se ve claramente. Después de haber recibido el mensaje http, se envían dos mensajes, esto se corresponde a los dos mensajes enviados en UDP. Después de esto se ven varios mensajes de HTTP, TCP y UDP, hay que entender que lo que hacen los actores es establecer una conexión, asique por eso se envían varios mensajes sucesivos, pero lo importante de esta captura es ver que se reciben mensajes de los tres protocolos.

Como ya se ha dicho antes, la captura en la que se ve con más detalle el intercambio de mensajes es la que se muestra a continuación:

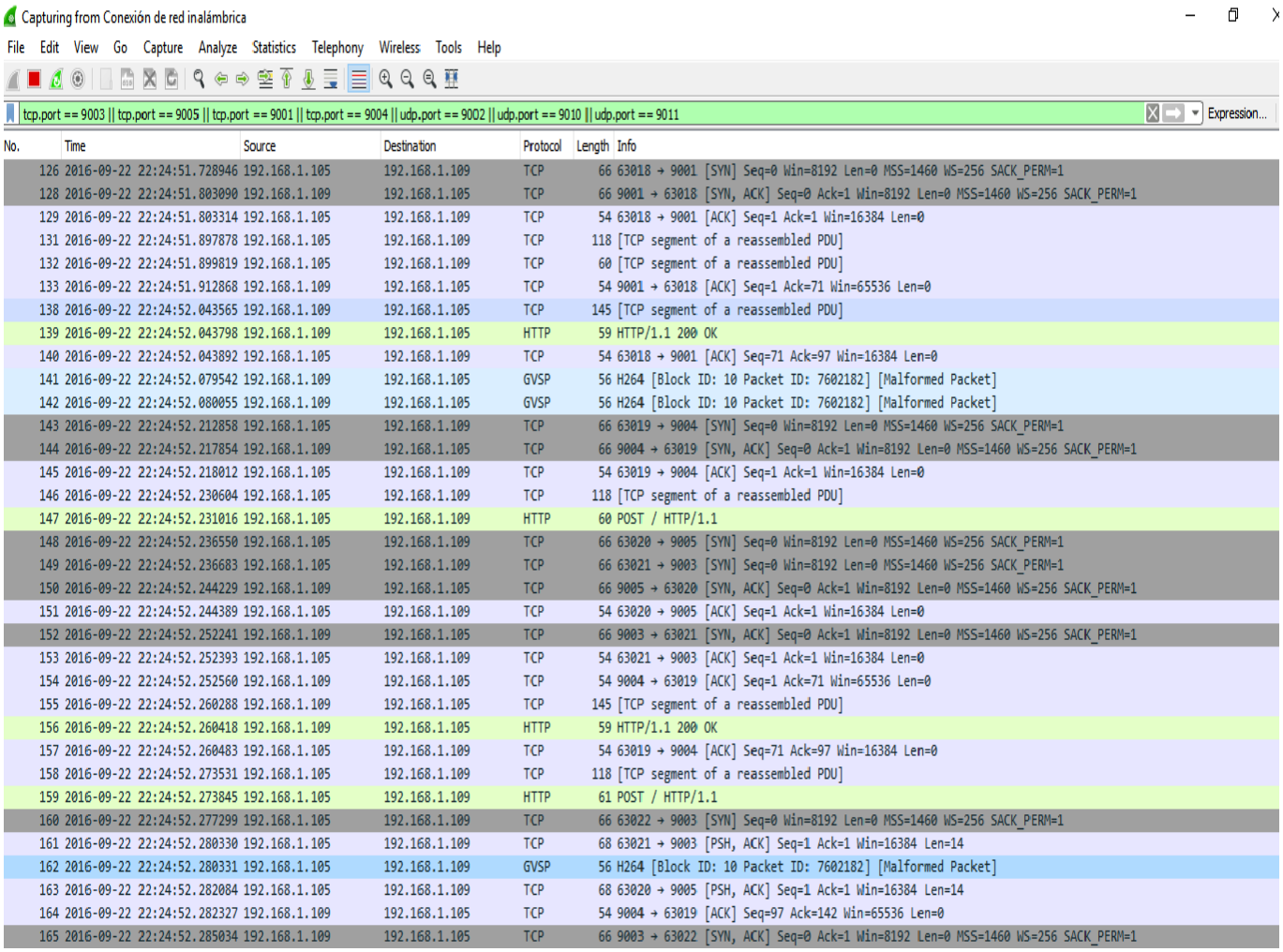

Figura 14: Captura de tráfico con WhireShark

Se comienza a analizar la captura anterior, donde se muestra todo el tráfico que nos interesa ver. Poco a poco se irá viendo cada mensaje y sus campos para ver que están correctamente según lo planeado.

Antes de nada hay que mencionar que la IP relacionada con el PC1 es 192.168.1.109, y la IP relacionada con PC2 es 192.168.1.105. Por otra parte, el mensaje que va al puerto 9001 del PC2 es la conexión HTTP inicial. Las dos conexiones UDP siguientes son las que van a los puertos 9010 y 9011. Por último, hay cuatro conexiones, dos de TCP, que van a los puertos 9003 y 9005, una de HTTP que va al puerto 9004 y una de UDP que va al puerto 9002.

Bien, el 126, es un mensaje de TCP del PC1 al PC2 en el puerto 9001, es con el que se inicia la conexión TCP para poder enviar el mensaje HTTP que se enviará del 131 al 138. En el mensaje 139 se puede ver la respuesta 200 OK al mensaje HTTP. A continuación, en los mensajes 141 y 142 se ven dos mensajes GSVP que tienen los siguientes campos:

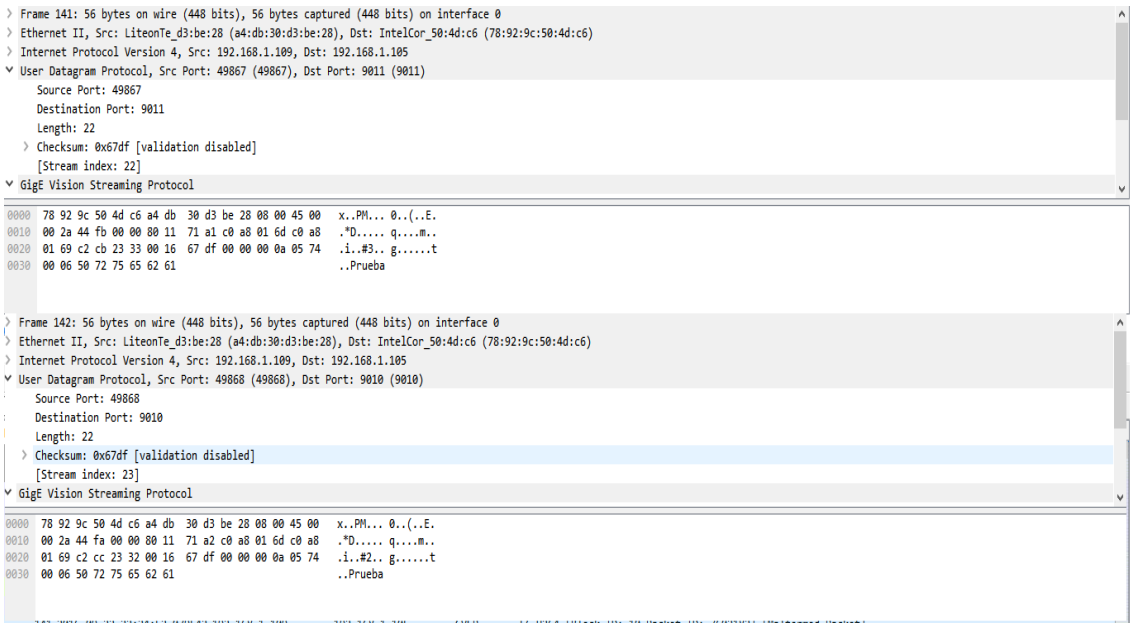

Figura 15: Mensajes UDP del PC2 al PC1

Como se puede ver en esta captura, se han enviado dos mensajes UDP, los de numero de 141 y 142, uno al puerto 9010 y otro al puerto 9011, tal y como se había dicho antes. Además podemos ver que los últimos caracteres del datagrama forman la palabra Prueba, es decir, el mensaje que hemos enviado.

Lo que tiene que hacer el PC1 al recibir los dos mensajes UDP, es enviar dos mensajes TCP, sobre los puertos 9003 y 9005; uno UDP en el 9002 y otro HTTP en el puerto 9004. Volviendo a la figura 14, en el mensaje 143 se inicia la conexión TCP sobre la que se enviará el mensaje HTTP, que como se ha dicho antes y se puede ver en la figura, se hace sobre el puerto 9004 del PC2. En el mensaje 147, se hace un POST de HTTP donde se envía el mensaje de prueba.

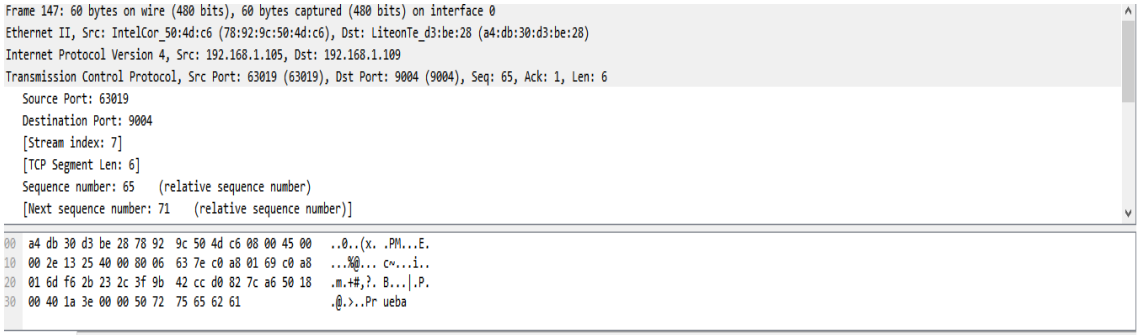

Figura 16: Datagrama 147, POST de PC1 a PC2

Como se ve en la figura, el POST va sobre el puerto 9004, y al final del mensaje incluye la cadena Prueba, que es la que se había enviado inicialmente.

En los mensajes 148 y 149 se inician las dos conexiones TCP sobre los puertos 9003 y 9005. Por último, en el mensaje 162, se inicia la conexión UDP sobre el puerto 9002, que es el último de los mensajes que quedaban por analizar, y en la figura que se ve a continuación se puede ver cómo se envía en el puerto indicado e incluye la cadena de prueba.

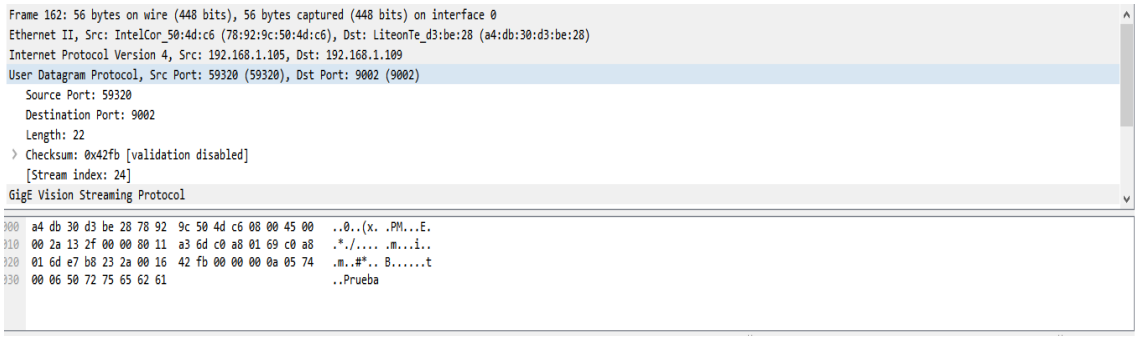

Figura 17: Datagrama 162

Como conclusión de este punto de prueba, se puede decir que los mensajes se han enviado correctamente según lo planeado. Queríamos enviar un mensaje en HTTP, luego dos en el otro sentido en protocolo UDP, y luego cuatro en el sentido inicial, siendo dos en protocolo TCP, uno en UDP y otro en HTTP. Se envían muchos más mensajes de los planeados, dado que Camel establece conexiones, no solamente envía los mensajes, lo cual dificulta levemente la visualización de los mensajes, pero gracias a eso se facilita en gran medida la transmisión de datos, ya que es totalmente transparente el protocolo que se use, y tan solo hay que preocuparse de los mensajes y del contenido.

Capítulo 7: marco regulador

El mercado de las telecomunicaciones ha sido uno de los más importantes en las últimas décadas. En cada sector dentro de éstas, hay múltiples instituciones de regulación. Son importantes los aspectos de seguridad, privacidad, fiabilidad, entre otros, y regular que se cumplan es competencia de dichas instituciones.

El internet de las cosas es uno de los sectores más novedosos en cuanto a telecomunicaciones, es un importante objetivo en cuanto a inversión. Se prevé que en los próximos años aumente exponencialmente, tanto el número de dispositivos, como de clientes, como de empresas que participan en este mercado. Por tanto es importante plantear si debería de necesitar cierta regulación o no, y en caso de tenerla, cómo llevarla a cabo. Pero debido a que es un sector muy novedoso, que lleva pocos años de investigación, no hay regulación actualmente.

Hay dos puntos de vista en este aspecto, por una parte una regulación excesiva o mal realizada frenaría el desarrollo de este sector. Pero por otra parte es importante que una tecnología que estará presente en la mayoría de los objetos o cosas que tengamos a nuestro alrededor, desde sensores de aparcamiento hasta sensores de tensión cardíaca, estén protegidos, estén regulados. Según José Otero, director de 5G Américas para el Caribe y América Latina, “El mundo está cambiando rápidamente hacia un ambiente en el que un número creciente de máquinas, como cámaras de vigilancia, redes de suministro eléctrico, automóviles, sensores hogareños e industriales, entre otros, estará conectado sin necesidad de la mediación de las personas. Esta transición afectará no sólo a las telecomunicaciones en sí misma, sino a un gran número de diferentes mercados verticales, economías, y esferas de la vida humana. El riesgo de que una regulación excesiva, o pobremente diseñada, podría desacelerar las enormes oportunidades de crecimiento de IoT en la región. Por ese motivo, es necesario que todo el ecosistema de actores dialogue y colabore con los reguladores acerca de este importante avance tecnológico”.

La regulación en las tecnologías, y especialmente ésta, es algo fundamental. Se puede regular desde el ancho de banda que tendrá cada dispositivo, hasta impulsar la competencia en el mercado, por tanto es importante que se tomen

medidas correctas en este aspecto.

Ian Brown, doctor del instituto de internet de Oxford, universidad de Oxford, debate en el documento de discusión del GSR15 (global syposium for regulators) sobre las implicaciones de IoT en las empresas, personas y sociedad, y pone especial atención sobre la regulación de las instituciones en este sector.

El documento explora las implicaciones regulatorias de la IoT para la concesión de licencias, la gestión del espectro, la normalización, la competencia, la seguridad y la privacidad. Normalmente las instituciones regulan sobre temas como privacidad, la competencia, la protección de datos, etc. Por otra parte, Un grupo de trabajo de expertos de la Comisión Federal de Comunicaciones de los Estados Unidos, predice que la mayoría de las comunicaciones en IoT funcionarán bajo tecnologías ya existentes, como Wi-Fi o 4G, asique no necesitarán espectro adicional para su funcionamiento.

Las empresas están a favor de que los estados no influyan por ahora en el internet de las cosas, y así, poder fabricar dispositivos sin necesitar licencias y a un coste más barato.

Como conclusión, se podría decir que en cuanto al internet de las cosas no hay ninguna regulación por el momento, y se debate si establecerla para garantizar una calidad de servicio en cuanto a seguridad o fiabilidad o mantener este sector sin regular por el momento, para promover su desarrollo y la facilidad de fabricación. La única regulación relativamente vigente, es la normalización en M2M, es decir, la normalización de la comunicación entre dispositivos. Los organismos que realizan dichas normas son principalmente ETSI, IEEE y oneM2M.

Capítulo 8: entorno socio-económico

La telecomunicación es uno de los mercados más emergentes en la actualidad. La ITU define[12] al periodo tecnológico desde el año 1971 aproximadamente hasta la actualidad como la era de las tecnologías de la información y la comunicación, la quinta revolución tecnológica.

Desde esta época, ha habido constantes cambios y evoluciones de las tecnologías existentes: La televisión cada vez da más servicios, la telefonía mejora, aparece la telefonía móvil, los microchips, la informática, entre otros avances. Actualmente, uno de los mercados que más emergen es el del internet de las cosas[13], incluso se llega a considerar como la siguiente revolución tecnológica.

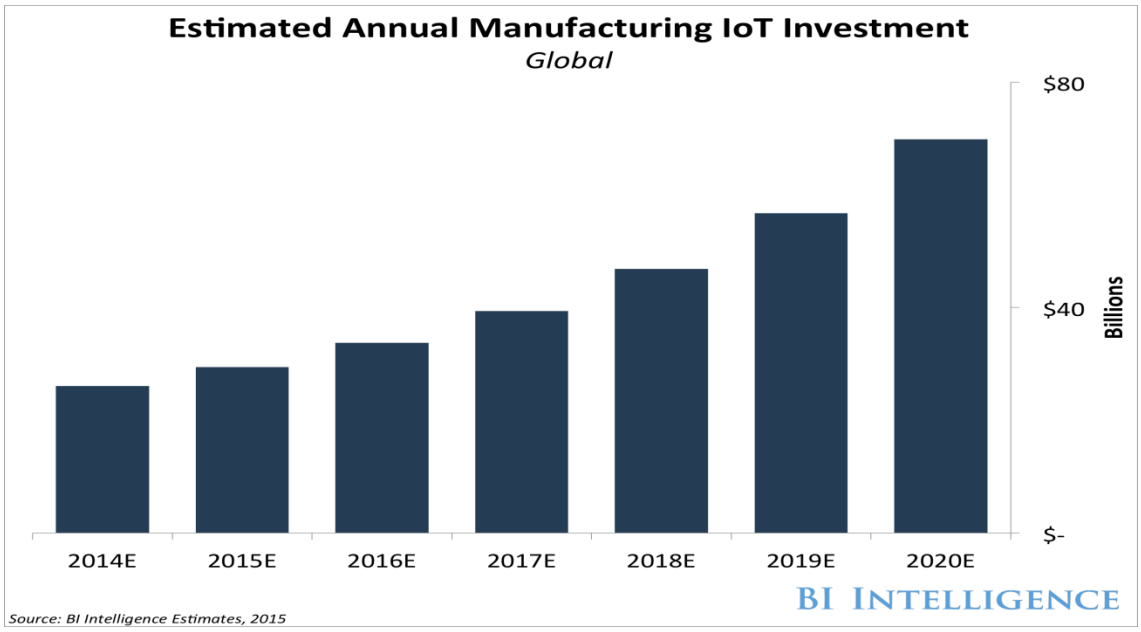

Figura 18: Estimación anual de inversión en IoT[14]

[12] Cristian Gomez 2012: "Revoluciones Tecológicas" en *Estado de la migración a Televisión Digital Terrestre a nivel mundial*

[13] John Greenough 2016: *How the 'Internet of Things' will impact consumers, businesses, and governments in 2016 and beyond*

[14] BI intelligence estimates, 2015

Como se puede ver en la figura anterior, se prevé que en los próximos haya un gran aumento de inversión en las tecnologías IoT. Es un sector por el que las empresas tecnológicas apuestan y que tiene gran futuro. Cuando estén más desarrolladas y se hagan más populares, seguramente sean socialmente muy populares, debido a la gran utilidad y diversidad que pueden alcanzar. Tanto en seguridad, como en conocimiento y comodidad, las tecnologías IoT serán eficaces y con el paso del tiempo llegarán a establecerse como elemento fundamental en las TICs.

En resumen, Este proyecto, completamente vinculado con las comunicaciones máquina a máquina y al IoT, está orientado a un sector que crece cada año. Las inversiones crecen, y los dispositivos por tanto también, y gracias a la utilidad que pueden tener, será esta tecnología algo que con el paso del tiempo se asiente en la sociedad, como en su tiempo fue la televisión o la telefonía.

Capítulo 9: presupuesto

Es importante a la hora de realizar un proyecto, conocer el presupuesto que costará llevarlo a cabo. Para ello, se debe hacer una estimación de lo que costará llevar a cabo todo el apartado del diseño.

Los gastos en material son dos ordenadores, y cinco servidores para el alojamiento del programa. Los dos ordenadores serán para que realicen el proyecto los dos programadores que se contratarán, y los servidores, serán para tener bastante capacidad de almacenamiento y procesamiento para permitir conectar múltiples dispositivos. Por cada servidor, una interfaz de ZIgBee, de Wi-Fi y de LTE. En cuanto al gasto en personal, se necesitará un director de proyecto, un analista, para revisar las normativas tanto técnicas como de mercado; un diseñador, que hará un esquema de cómo debe ser el proyecto; y dos programadores para realizar el software.

COSTES MATERIALES				
Concepto	**Coste (€)**	**Dedicación (meses)**	**Periodo de depreciación (meses)**	**Coste imputable (€)**
Ordenadores x2	500	6	24	250
Servidores x5	200/año	24	-	2000
Interfaz ZigBee x5	60	24	36	300
Interfaz Wi-Fi x5	20	24	36	66.66
Interfaz LTE x5	30	24	36	100
Total				2716.66

Figura 19: Coste en material

El tiempo dedicado para el proyecto será de 6 meses, y el tiempo de depreciación según qué material sea, será de 2 o 3 años. El alquiler de servidores no tiene periodo de depreciación por no ser adquisición. El coste total se obtiene con la siguiente fórmula:

$$Coste\ imputable = \frac{Dedicación}{Periodo\ de\ depreciación}\ x\ coste\ total$$

COSTES EN PERSONAL			
Concepto	**Dedicación (horas)**	**Coste/hora (€)**	**Total (€)**
Director	160	25	4800
Analista	240	10	2400
Diseñador	240	15	3600
Programador x2	480	13	12480
Total			23280

Figura 20: Coste en personal

Total	
Concepto	**Coste**
Material	2716.66
Personal	23280
Total	25966.66

Figura 21: Coste total

Capítulo 10: Conclusiones y líneas de trabajo futuras

Llegamos al punto final del Proyecto, donde se repasarán los objetivos propuestos y su grado de satisfacción, además de los problemas encontrados, y como último apartado los proyectos que se podrían realizar para continuar este trabajo.

El proyecto ha tratado de un análisis de la tendencia actual al desarrollo de aplicaciones IoT y por tanto, a las tecnologías M2M. A la vista de esta situación se ha decidido diseñar un Gateway que pueda unir dispositivos, y además, en gran cantidad. El diseño engloba determinados aspectos como la escalabilidad, la variabilidad de protocolos en diferentes niveles, tanto de transporte como de enlace, o el desarrollo de una interfaz para poder crear más fácilmente las conexiones con dispositivos. La implementación, ha sido el desarrollo de un pequeño programa que muestre con determinadas tecnologías es posible realizar un elemento que tenga esta funcionalidad.

A continuación se enumeran algunos de los puntos positivos del desarrollo de este proyecto: desde el punto de vista personal, he aprendido a usar nuevas tecnologías que facilitan muchísimo el trabajo, como Camel a la hora de usar protocolos reales en las conexiones. Por otra parte ha sido interesante realizar un programa que pueda ejecutarse en diferentes máquinas y que podamos programar una comunicación entre dichas máquinas. También es interesante haber hecho un Gateway, que perfectamente podría usarse como un traductor, como mezclador, como proxy, ya que podemos abrir los mensajes que vayan por la red, coger su contenido, juntarlo con otros mensajes, cambiarles el protocolo, etcétera.

Respecto a los puntos negativos, cabe destacar que el programa ha quedado en bastante menos de lo que en el diseño se había planteado. Los conocimientos de programación limitan las posibilidades a la hora de realizar el programa, como por ejemplo, la interfaz gráfica. Por otra parte, los medios también limitan, puesto que habría sido interesante haber implementado además de los protocolos de encima del nivel de red, haber implementado los protocolos de

enlace, como el Wi-Fi, el ZigBee, el LTE, entre otros que podrían tener mucha utilidad. En definitiva, los puntos negativos vienen de la implementación, pues ha quedado corta comparada con el diseño, aunque cumple con los objetivos establecidos inicialmente, ya que desde el principio el programa no pretendía cumplir todas las expectativas sino tan solo comprobar que estas tecnologías funcionarían.

En cuanto a lo original del proyecto, se puede destacar el programa implementado, puesto que ha sido totalmente desarrollado de cero, salvo alguna pequeña parte que ha sido obtenida de la documentación de las herramientas empleadas.

10.1 Problemas encontrados

En el diseño, teórico, se ha intentado planificar qué se hará en cada parte del proyecto, pero a la hora de implementarlo, siempre surgen problemas con los que no se contaban. A lo largo del desarrollo del proyecto ha habido determinados inconvenientes, ninguno de gravedad, pero si problemas que no han permitido realizar el programa como se tenía planeado en un primer momento.

Los principales problemas que ha habido durante el proyecto, han sido principalmente a la hora de desarrollar los actores de los diferentes protocolos. Inicialmente se tenían planeados ciertos protocolos como ZeroMQ, SMTP o HDFS, pero no han podido ser desarrollados. ZeroMQ es un protocolo usado para el envío de notificaciones, con un modelo de suscriptor-publicador, pero el problema es que la implementación que tiene camel de este protocolo es muy simple, debe ser complementada con un programa aparte realizado manualmente por el usuario, y se volvía muy complejo el uso de ZeroMQ en nuestro programa. Respecto a HDFS y SMTP, al implementarlos requerían conectar con un servidor especial de estos protocolos, y con el modelo que se quería implementar de consumidor-productor entre dos ordenadores normales no funcionaba como se esperaba. Por tanto a lo que se limitó esa parte del proyecto es realizar protocolos sencillos de intercambio de mensajes,

como HTTP, TCP y UDP, que son los que se han usado finalmente.

Respecto al resto del programa no ha habido ningún problema notable, como mucho fallos de incompatibilidades de versiones de las herramientas usadas, pero observando bien en el repositorio de maven cada librería, sus versiones, sus dependencias, etcétera, se han podido encajar las versiones de tal forma que todo lo que se usa de cada herramienta pueda seguir usándose y combinándose con las otras librerías.

Como última nota de este punto, por mencionar una dificultad fuera de la programación y el diseño del proyecto, sino en el área de la investigación, cabe destacar la búsqueda de normativa del sector de las tecnologías M2M, dado que es una parte de las telecomunicaciones que aún no está del todo desarrollada, que está en proceso de investigación, es complicado encontrar normativas y regulaciones por parte de las instituciones de estas tecnologías. Como se mencionó al concluir el apartado 6 (marco regulador), no hay realmente una normativa en cuanto al mercado del internet de las cosas, y respecto a las tecnologías tampoco hay nada totalmente establecido.

• 10.2 Líneas de trabajo futuras

Último apartado del capítulo conclusiones, aquí es donde se comentarán las posibles continuaciones que pueda tener este proyecto, o como mejorarlo en un futuro, pudiendo tener más medios y más tiempo.

Una de las ideas más interesantes aunque no es exactamente una continuación del proyecto, sería programar un Gateway con la misma funcionalidad y las mismas características de las planteadas en este proyecto, pero tratando de programarlo cumpliendo las normativas que las entidades de estandarización exigen. Se ha comprobado en este proyecto que las herramientas empleadas pueden ser muy útiles, muy sencillas y muy seguras a la hora de emplearlas para conectar dispositivos, pero no se ha tenido en cuenta el realizar la programación de estas conexiones de una determinada forma que las instituciones

normalizadoras requieran. Eso es algo que en un futuro sería buena idea probar.

Otra posible prolongación del proyecto sería la de el desarrollo de dispositivos que hagan la función de puntos finales de una red de cosas. En este proyecto se ha comprobado el funcionamiento del programa usando conexiones y solicitudes generadas por el mismo ordenador o por otros ordenadores cercanos con los que se pudiera probar, pero no se ha tenido la posibilidad de usar dispositivos que se conecten al programa. Sería ideal tener varios miniPCs del estilo del Raspberry PI o Arduinos o similares, conectarlas con periféricos de Wi-Fi, de Ethernet, de LTE, y programarlas para que envíen datos al Gateway central y ver con un programa de captura de tráfico si se intercambian los mensajes entre los diferentes dispositivos.

Por último, cabe mencionar el apartado de la escalabilidad, ya mencionado a lo largo del proyecto pero que debido a los medios no es posible llevarlo a cabo. Sería muy útil modificar el programa para que funcionara en una red de ordenadores, así se podría aumentar las características de procesamiento y almacenaje, o el número de conexiones máximas.

Chapter 11: Conclusions and future research

We reached the end point of the project where the proposed objectives and their level of satisfaction will be reviewed, along with the problems encountered, and as a last section projects that could be made to continue this work.

The project has attempted an analysis of the current development trend of IoT applications and therefore the M2M technology. In view of this situation it was decided to design a gateway that can link devices, and also in large quantities. The design includes certain aspects such as scalability, variability of protocols at different levels, both transmission and link, or the development of an interface that facilitates the creation of connections to devices. The implementation has been the development of a small program that shows certain technologies can make an element that has this functionality.

Below are some of the pluses of the development of this project are listed: from the point of view, I learned to use new technologies that greatly facilitate work, such as Camel when using real protocols in the connections. On the other hand it has been interesting to have a program that can run on different machines and can schedule communication between the machines. It is also interesting to have made a Gateway, that could easily be used as a translator, as a mixer, as proxy, as we can open the messages that go through the network, catch your content, put it together with other messages, change their protocol, and so on.

Regarding the negative points, it should be noted that the program has been in considerably less than that the design had been raised. Programming skills limit the possibilities when implementing the program, such as the graphical interface. Moreover, the media also limited, since it would have been interesting to have implemented in addition to the protocols above the network level, have implemented the link protocols, such as Wi-Fi, ZigBee, LTE, among others that could be very useful. In short, the negatives come from the

implementation, as has been short compared to the design, but meets the objectives set initially, because from the beginning the program was not intended to meet all expectations but only verify that these technologies would work.

As for the original project, you can highlight the program implemented, since it has been completely developed from scratch, except for some small part that has been obtained from the documentation of the tools used.

11.1 Problems found

In the design, theoretical, it has tried to plan what will be done in each part of the project, but when it comes to implementing it, always problems arise that were not counted. Throughout the development of the project there have been certain drawbacks, none seriously, but if problems have not allowed implement the program as planned at first.

The main problems that occurred during the project have been mainly in developing players in the different protocols. Initially they had planned ZeroMQ certain protocols such as SMTP or HDFS, but have not been developed. ZeroMQ is a protocol used for sending notifications, with a model of subscriber-publisher, but the problem is that implementation that has camel of this protocol is very simple, must be complemented by a separate program manually by the user, and it became very complex ZeroMQ use in our program. Regarding HDFS and SMTP, required to implement a special server to connect to these protocols, and the model that wanted to implement consumer-producer between two normal computers did not work as expected. So what that part of the project is dedicated to make simple message exchange protocols, such as HTTP, TCP and UDP, which are those who have finally used.

The rest of the program there has been no noticeable problem, as much failures incompatibilities versions of the tools used, but looking good in the repository maven each library, its versions, dependencies, etc., have

been fit versions so that all that is used every tool can still be used and combined with other libraries.

As a final note of this point, to mention a difficulty outside the programming and design of the project, but in the area of research, include the search for regulations of M2M technology sector, since it is a part of telecommunications which is not yet fully developed, which is under investigation, it is difficult to find rules and regulations by the institutions of these technologies. As mentioned at the end of section 6 (regulatory framework), there is really no rules as to market the internet of things, and on the technologies there are also no fully established.

• 11.2 Future Research

Last section of chapter conclusions, this is where the possible continuations you may have this project or how to improve it in the future, being able to have more resources and more time will be discussed.

One of the most interesting ideas while not exactly a continuation of the project, it would set a Gateway with the same functionality and the same characteristics as those raised in this project, but trying to program compliance regulations that require standardization bodies. It has been found in this project that the tools used can be very useful, very simple and very safe when using them to connect devices, but has not been taken into account to program these connections in a certain way that institutions normalizing required. That is something that in the future would be a good idea to try.

Another possible extension of the project would be the development of devices that make the function endpoints of a network of things. This project has verified the operation of the program using connections and requests generated by the same computer or other nearby computers with which it could be proved, but has not been able to use devices that connect to the program. It would be ideal to have several miniPCs style Raspberry PI or Arduinos or the like, connect with peripherals WiFi, Ethernet, LTE, and schedule them to send data to the central Gateway

and see with a program to capture traffic, if messages are exchanged between different devices.

Finally, we should mention the section of scalability, mentioned throughout the project but because the media is not possible to carry it out. It would be useful to modify the program to function on a computer network and could increase processing and storage characteristics, or the maximum number of connections.

Referencias

[1] Akka documentation: introduction. What is Akka?
[2] Akka documentation: general. What is an actor?
[3] Akka documentation: Event Bus
[4] Apache Maven: About Maven.
[5] Apache Camel: index
[6] Ibsen, Claus; Anstey, Jonathan; "Camel in Action", Manning Publications, 2010
[7] Iyappan Ramachandran, “A deeper look at LTE”, Agilent Technology, 2010
[8] Balazs Bertenyi, “3GPP LTE Standards Update”, 2012
[9] Dr. Ian Brown, GSR discussion paper: “Regulation and the internet of things”, 2015.
[10] ITU news: “Regulación y la internet de las cosas”, 2015.
[11] Diario Ti: “5G Américas pide regulación sobre IoT”, 2016.
[12] tyniot: “Reguladores de Europa abordan Internet de las Cosas y regulación OTT”
[13] Abogacía española: “Algunas reflexiones sobre el Internet de las Cosas y las comunicaciones máquina a máquina”
[14] Conner, Margery (27 de mayo de 2010). Sensors empower the "Internet of Things" pp. 32-38
[15] P. Magrassi, A. Panarella, N. Deighton, G. Johnson, “Computers to Acquire Control of the Physical World”, Gartner informe de investigación T-14-0301, 28 Septiembre, 2001
[16] Gershenfeld, Nel, Raffi Krikorian y Danny Cohen, "The Internet of Things" Scientific American, octubre 2004, p. 79
[17] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle, “From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence”, Elsevier, 2014, ISBN 978-0-12-407684-6
[18] ABI Research, “More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything” in 2020
[19] Torres, Álvaro. Telecomunicaciones y telemática. De las señales de humo a las redes de información y a las actividades por internet. Tercera edición: 2007, Colombia, Colección Telecomunicaciones.
[20] ZigBee Alliance. (2012). ZigBee Specifications

[21] WiMAX Forum. (2012). WiMAX Forum Withe papers
[22] WinWin Magazine, January 2010,” M2M: The Internet of 50 Billion Devices".
[23] HowStuffWorks.com, “How Machine-to-Machine Communication Works".
[24] Pervasive Computing Laboratory, “Introduction to IoT, Smart Cities and related concepts”.

Anexo I: planificación

Actividades	Actividades antecedentes	Duración (semanas)
A: investigación del sector M2M	-	2
B: análisis de normativas	A	1
C: análisis del entorno socio-económico	A	1
D: diseño	A	4
E: elección de tecnologías	D	0.5
F: determinación de módulos a realizar	D	1
G: desarrollo e implementación	E,F	6
H: Fase de pruebas	G	1

Figura 22: Tabla de actividades, con antecedentes y duraciones

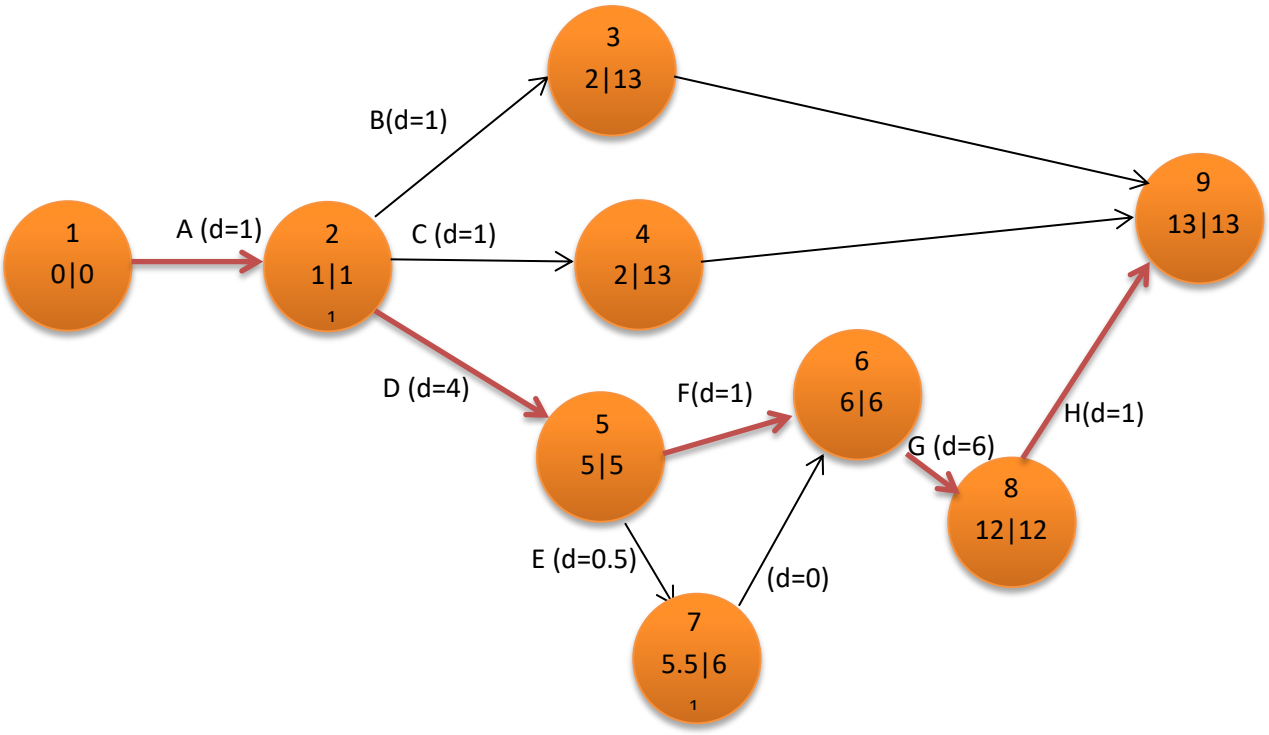

Figura 23: diagrama PERT de las actividades